Flexible and Fault-Tolerant Communication for Safety Critical Real-Time Systems

Fawad Riasat Raja

B.Sc. Software Engineering (Hons.)

M.Sc. Computer Engineering

School of Information and Communication Technology
Griffith University
Australia

SUBMITTED IN FULFILMENT OF THE REQUIREMENTS OF THE DEGREE OF DOCTOR OF PHILOSOPHY

April 2022

Abstract

In distributed real-time systems, control is distributed over a number of nodes connected in a network. Some nodes rely on information provided by other nodes as an input to perform their operations. Therefore, distributed real-time systems heavily depend on their communication subsystems, which are responsible for timely and error-free delivery of the information among distributed nodes. A real-time system is required to provide its services within a defined time frame, which means it must meet certain deadlines. Safety-critical real-time systems such as fly-by-wire, drive-by-wire and the like, must not only meet the criteria of timeliness, but also must be fault-tolerant, as missing a deadline or a fault may have catastrophic consequences. At the same time, engineering real-time communication is a complex task, particularly in the safety-critical domain. To make the complexity manageable when designing dependable systems in the safety-critical domain, these systems are traditionally kept simple, rigid, and inflexible. This is increasingly becoming a challenge, as hardware and software are becoming more and more capable, and is used in systems that are becoming more and more autonomous. Examples range from advance driver assistance systems to self-driving cars. This poses a significant challenge for communication protocols that exists in this domain.

The Time-Triggered Architecture (TTA) guarantees timeliness in all circumstances by requiring communication scheduling ahead of time, and hence, it is used to produce more dependable real-time systems. TTP/C and FlexRay are two widely used TTA based bus protocols for safety-critical real-time systems. In this thesis, I have investigated the impact of the flexibility issues of existing Time-Triggered (TT) systems through a number of case studies such as brake-by-wire and an autonomous vehicle. I have demonstrated the lack of flexibility results in poor bandwidth and channel utilisation. Following that, I have analysed the shortcomings of existing protocols in a systematic way. Based on this analysis, I propose a protocol that is suitable for modern, autonomous safety-critical real-time systems and provides the flexibility needed for the complex payload requirements of these systems. My proposed approach uses flexible communication schedules based on the transmission payload requirements of participating nodes not only in a single Time Division Multiple Access (TDMA) round but also over multiple TDMA rounds of a cluster cycle. Multiple operational modes are also supported by allowing each mode to have a different communication schedule as per transmission requirements. As with all safety-critical communication, it is vital to prevent the communication channel from being monopolised by a faulty node. The existing fault-tolerant model to tackle such faults cannot work with the proposed approach due to its flexible communication schedules. I therefore investigate approaches for bus guardians that prevent such monopolisation. I propose a system of guardian nodes that are fully aware of the assigned communication schedules and the current situation by listening to the traffic of the channel. My analysis shows that these guardians will block any faulty node trying to transmit outside its assigned timing window. This thesis presents the formal verification model to verify the timing parameters of participating nodes such as transmitter, receiver, and guardian nodes. To tackle the design and implementations issues, I have used a model-driven engineering approach that utilises a high-level design of verifiable and directly runnable implementations. A subsumption architecture with clear execution semantics is used to implement the more complex system behaviours at a high level, mitigating the complexity of state replication. This subsumption architecture made it possible to incrementally refine the implementation without interfering with unaffected components of the system.

The proposed approach improves the channel utilisation by allowing the slot length of nodes to be configured in accordance with the actual payload requirements of these nodes inside a TDMA round. While this approach is based on the traditional TDMA scheme utilised in the TTA, it significantly improves bandwidth utilisation over the traditional schemes. The analyses performed in this thesis have shown that gross overhead time is reduced by almost 90%, improving overall bandwidth utilisation efficiency almost twofold in a typical automotive brake-by-wire system scenario. Furthermore, channel utilisation is also increased by allowing the slot length of each node to be configured in accordance with its actual transmission payload requirements for each TDMA round of a cluster cycle. This eliminates node slot idle times for all nodes, hence reduces transmission overhead. This flexibility makes it possible to reduce the gross overhead time by almost 99%, improving overall bandwidth utilisation efficiency almost nine times compared to existing TTA-based communication protocols in an autonomous vehicle system case study. Despite the added flexibility, the same level of predictability has been maintained. My approach not only increases flexibility and channel utilisation for safety-critical payload, but also maintains the ability to handle faults in a fail-silent way, at the same level as other TTA-based protocols.

Statement of Originality

This work has not previously been submitted for a degree or diploma in any university. To the best of my knowledge and belief, the thesis contains no material previously published or written by another person except where due reference is made in the thesis itself.

Signed: [redacted]

Fawad Riasat Raja

Acknowledgements

First of all, I would like to thank Allah, The Almighty for listening to my prayers and keeping me safe. He provided me with opportunities, and gave me determination and strength to do my research. His continuous Grace and Mercy throughout my life made things easy and even more during the tenure of my research.

I truly acknowledge that the prolific work presented in the thesis has become possible only with the support, guidance and encouragement of my supervisors. I would also thank my friends and family for their constant backing whenever I felt down. Their assistance made this knowledgeable and arduous journey possible.

I would like to express my sincere gratitude to my principal supervisors, Dr. Rene Hexel and Dr. David Chen. Their continuous support, patience, motivation, enthusiasm and immense knowledge made my PhD research a doable feat. Their constructive feedback showed me the way to build my knowledge-base for the subject research. Dr. Hexel and Dr. Chen mentored an amicable and positive disposition, and have always been available to explain my confusions and misunderstandings. Their sagacious discussions helped me to understand various aspects of research problems. Over the course of my research journey, I considered them to be my good friends with whom I was comfortable to share my personal problems as well. They also put themselves in my shoes when a setback struck my research. They fully cooperated during the period when I was going through multiple surgeries owing to Lisfranc injury. Thank you Dr. Hexel and Dr. Chen, for your tremendous support. I am strongly optimistic about future research collaboration with your kindness.

Furthermore, I would extend my gratitude to all the scholars of *Machine Intelligence and Pattern Analysis Laboratory* (MiPAL). It would be unfair if I don't express my sincere gratitude to the heads of *School of Information and Communication Technology*, who have been appointed during my stay at Griffith, for motivating me to work for the betterment of the school, and providing a pleasant environment for PhD studies. A special thanks to the present Head of School, Professor Paulo de Souza, for his various meetings that raised my morale. I am also indebted to administrative staff of *School of Information and Communication Technology* and *Griffith Graduate Research School* of the Griffith University for their assistance.

Contents

List of Figures

List of Tables

Abbreviations

ABS	**A**nti-lock **B**raking **S**ystem
ACL	**A**ccess **C**ontrol **L**ist
ADAS	**A**davnced **D**river **A**ssistance **S**ystem
AVB	**A**udio **V**ideo **B**ridging
BAN	**B**rake **A**ctuator **N**ode
BBWM	**B**rake **B**y **W**ire **M**anager
BG	**B**us **G**uardian
BPN	**B**rake **P**edal **N**ode
CAN	**C**ontroller **A**rea **N**etwork
CC	**C**luster **C**ycle
CNI	**C**ommunication **N**etwork **I**nterface
CRC	**C**yclic **R**edundancy **C**heck
CSMA	**C**arrier **S**ense **M**ultiple **A**ccess
CState	**C**ontroller **State**
CU	**C**hannel **U**tilization
ET	**E**vent **T**riggered
FTA	**F**ault **T**olerant **A**verage
FTAM	**F**ault **T**olerant **A**lgorithims and **M**echanisms
GCL	**G**ate **C**ontrol **L**ist
GOPs	**G**roup **O**f **P**ictures
LLFSM	**L**ogic **L**abelled **F**inite **S**tate **M**achines
MARS	**M**aintainable **A**rchitecture for **R**eal-time **S**ystems
MEDL	**M**essage **D**escriptor **L**ist
NGU	**N**ever-**G**ive **U**p

NIT	**N**etwork **I**dle **T**ime
NSCI	**N**on-**S**afety **C**ritical **I**nformation
PAR	**P**ositive **A**cknowledgement and **R**etransmission
PTP	**P**recision **T**ime **P**rotocol
SAE	**S**ociety of **A**utomotive **E**ngineers
SCI	**S**afety-**C**ritical **I**nformation
SCRT	**S**afety-**C**ritical **R**eal-**T**ime
SFN	**S**ensor **F**usion **N**ode
SG	**S**ynchronisation **G**ap
SW	**S**ymbolic **W**indow
TAS	**T**ime-aware **S**haper
TCP	**T**ransmission **C**ontrol **P**rotocol
TG	**T**erminal **G**ap
TI	**T**ransmit **I**nterval
TSN	**T**ime **S**ensitive **N**etworking
TT	**T**ime **T**riggered
TTA	**T**ime **T**riggered **A**rchitecture
TTP	**T**ime **T**riggered **P**rotocol
WCET	**W**orst **C**ase **E**xecution **T**ime
WSSN	**W**heel **S**peed **S**ensor **N**ode

Symbols

T^{trad_r}	TDMA round in the traditional slot allocation approach
T^{inc_r}	TDMA round in INCUS
τ^{max}	The node slot length in the traditional slot allocation approach
τ_i^{inc}	Slot length for each node in INCUS (possibly different for each node i)
T_i^{trans}	Transmission time for control and payload data for node i
T_i^{idle}	Slot time for node i not utilised for data or control information
T^{ifg}	Time when no transmission occurs between frames
T_i^{ovhd}	Total overhead for node i for its allocated slot
$Slot_{st}$	Time to start the node slot
$Slot_{len}$	Length of the node slot
F_tS	Maximum time to start frame transmission
F_tE	Time to end frame transmission
GS	Time to open bus guardian window
GB	Time to block the transmission
GE	Time to shut bus guardian window
R_wS	Time when receiver open its window to receive the frame
R_wE	Time when receiver closes its window
T^{fr_r}	Length of FlexRay TDMA round (where $T^{fr\_r_m} = T^{fr\_r_n}$)
T^{inc+_r}	Duration of TDMA round in INCUS+
τ^{max}	Slot length of each node in FlexRay approach
τ^{inc+}	Slot length of each node in INCUS+
$\hat{T}^{ovhd\_r}$	Total overhead time in a TDMA round
$\hat{T}^{ovhd\_c\_cycle}$	Total overhead time in a cluster cycle

$T^{fr\_c\_cycle}$	Total length of cluster cycle in FlexRay slot allocation approach
$T^{inc+\_c\_cycle}$	Total length of cluster cycle in INCUS+ approach

Chapter 1

Introduction

Real-time systems are gaining greater importance in our daily life and we have seen an explosive growth of these systems over the last few decades. A real-time system is required to provide its services within a defined time-frame, which means it must meet certain deadlines [2]. There are many kinds of embedded real-time systems, e.g. command and control systems, telecommunication systems and the like [11]. These systems play a vital role by providing different services. For example, when we are driving, these systems are manage the control of the engine and handle the brakes of our car. When we are flying, these systems are control the landing, take-off, and other services essential for flying our plane. Real-time systems are also used to manage the smooth flow of traffic, e.g. by controlling the traffic lights. In medical care, these systems provide services to monitor our blood pressure and heartbeats [1, 2, 11]. In summary, we can say that our dependence on the use of real-time systems in our daily life is increasing day by day, and, as technology evolves, the systems are becoming increasingly more complex.

On the basis of their functional requirements real-time systems can be divided into two major types, safety-critical systems and non-safety-critical systems [2]. Safety-critical real-time systems such as fly-by-wire, drive-by-wire and the like, must meet the criteria of timeliness and must be fault-tolerant [2]. This is vital, as missing a deadline by the system or a fault in the system may have catastrophic

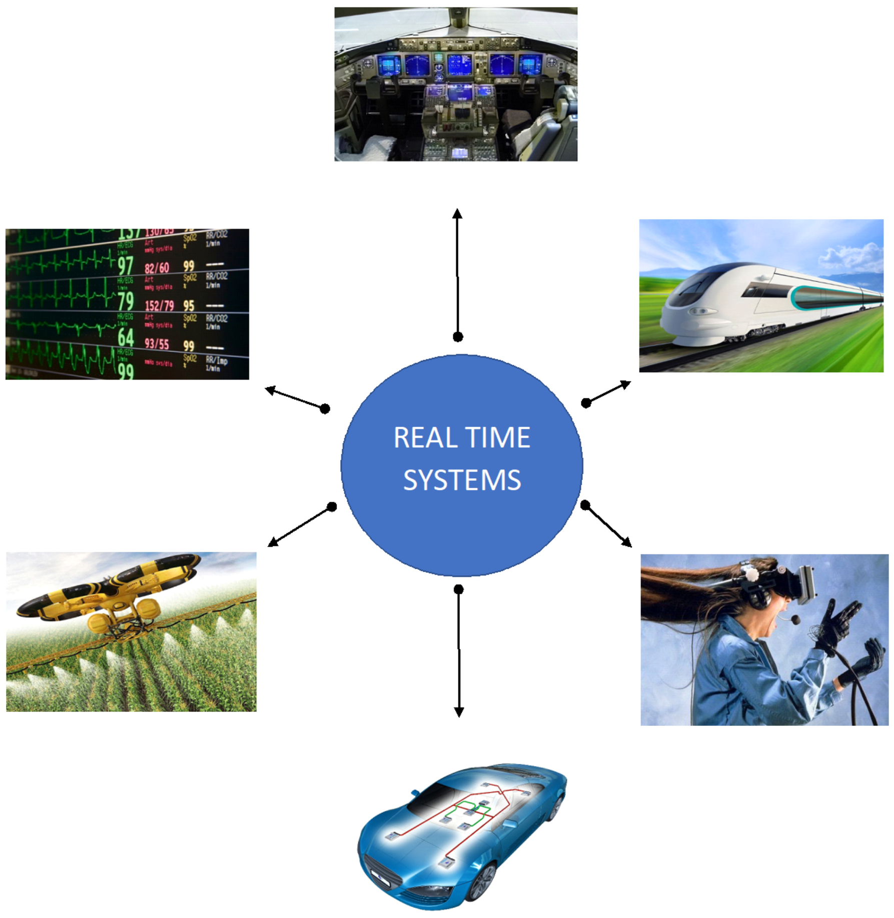

FIGURE 1.1: Real Time Systems [1]

consequences, including loss of life [2]. By contrast, missing a deadline in non-safety-critical real-time systems such as online audio/video streaming systems, can be an inconvenience, but does not have any catastrophic effect [2].

1.1 Challenges

In distributed safety-critical real-time systems, different nodes are connected with each other through a shared communication channel such as a bus. They coordinate their actions through message-passing, therefore, timely and reliable message delivery is critical. Communication errors and unpredictable delays in transmission may lead to unpredictable behaviour of distributed real-time systems [2]. For instance, consider a brake-by-wire system in a car. When the driver hits the brake pedal, then a brake force is calculated and transmitted to each wheel for stopping the car. Depending on the speed the car is traveling at, a slight delay or error in communication may lead to (some of) the brakes engaging too late, potentially causing harm [2].

These delays can have a severe impact, no matter at which communication layer they occur on. When nodes share single communication channel, generally only one node, at a time, is allowed to transmit over the communication channel [6]. Attempts of concurrent transmission from multiple nodes will often interfere with each other. This is known as a transmission collision and all the nodes involved in the transmission can lose their messages [2]. There are different strategies [12–14] that can be used to minimise the collisions over the communication channel. The most common one is sensing the channel before attempting to transmit and delay the transmission if the channel is busy [14].

Often non-safety-critical environments use Event-Triggered (ET) communication, where communication activities are initiated on the occurrence of significant events [12]. A large number of events are unpredictable in nature [12]. This means that a transmission activity from any node can take place at any time [12]. Various safety mechanisms [14][13] are utilised to handle errors in communication. In event-driven communication, error detection requires positive acknowledgments, as can be seen in the Transmission Control Protocol (TCP) [2]. In the positive acknowledgment mechanism, a transmitter node receives an acknowledgment from the receiver node if its transmission is successful [2]. This process requires a time-out that can accumulate to an indefinite amount of time if other nodes are causing interference on the communication channel [2].

In the case of responsive systems, e.g. a brake-by-wire system, these delays can compromise the usefulness and safety of the system [2]. As we will see in more details later, the point is that ET communication is unpredictable and is therefore unsuitable for responsive or safety-critical real-time systems.

In light of these issues, a new communication architecture, known as the Time-Triggered Architecture (TTA) was introduced by C. Scheidler et al. [6] and now it is widely used in industry. This architecture guarantees that communication that occurs within a system's specified fault hypothesis arrives within its specified deadline [6]. In the TTA, channel access is based on a Time Division Multiple Access (TDMA) scheme where communication time slots for nodes to access the channel are scheduled at design time [12]. All nodes use synchronised clocks and know exactly at what point in time each node will transmit its message, guaranteeing that channel access will be free of collision [8]. This Time-Triggered (TT) communication approach removes the necessity of explicit communication acknowledgments and provides predictable communication even in congested networks that are utilised at capacity [6, 8].

Despite these benefits, the *lack of flexibility* has been one of the biggest obstacles when it comes to the adoption of a TTA system. TTA systems follow strict periodic schedules that are configured offline [6]. Fault-tolerance services, such as membership and bus guardians complement these static schedules [8]. However, if the dynamics of communication patterns are changed over the time for safety-critical systems and to accommodate such requirements flexibility becomes a must requirement for these systems to work efficiently. Despite the significant research in this area, the challenge of flexibility while retaining predictability and fault-tolerance in safety-critical real-time system still needs to be addressed. The focus of this research is to overcome this fundamental challenge and design a flexible communication protocol for safety-critical real-time systems that can accommodate the flexibility needed for today's complex applications as well as changes in the environment, while ensuring the same, stringent level of reliability and safety. This thesis explores a number of real-world scenarios and analyses how the lack of flexibility can impact the essential requirements such as timeliness and fault-tolerance of safety-critical real-time systems.

The existing TTA based protocols for safety-critical real-time systems such as TTP/C, were designed at a low, procedural level and implemented using a mix of programming languages such as C++ and assembly [15]. This approach certainly offers predictable, high performance direct hardware implementation but a key limitation is the design and development effort, which may take to several man years, severely limiting what can be achieved. Moreover, such a low-level implementation is often wedded to a specific hardware, and therefore, is difficult to port to a different platform. The rigorous timing requirements for safety-critical real-time systems that need to be modelled early on in the design process has made it difficult to model verifiable executable real-time behaviour at a high level [16]. Another objective of the thesis is therefore to investigate how to overcome such design and development issues by using a high-level design approach using executable models to facilitate the incremental development of the protocol.

1.2 Research Question

Is it possible to architect communication for safety-critical real-time systems that supports flexibility without compromising safety and reliability?

After years of research in distributed real-time systems, this question still needs to be addressed. There are many types of distributed systems, but the focus of this thesis is on the safety-critical distributed real-time systems that interacting with real-world objects. Examples of such systems are assisted braking system in automotive, aircraft control, factory automation systems, etc. [17][18][2]. To eliminate single points of failure, these are designed as large-scale distributed systems that link multiple sensors with computers through bus or star network topology [2]. Importantly, such control systems are involved in controlling real world objects, therefore, they have timing constraints dictated by the physical characteristics of the environment and the objects they control [2]. Consequentially, it is necessary for the communication subsystems used in such control systems to handle the associated constraints in communication [19]. By far, bounded latency is considered one of the most important factors in real time communication [20] [21]. The timing constraints are defined through deadlines that determine the correct temporal behaviour of safety-critical real-time systems [2]. Other factors fundamental to the correctness of such systems include bandwidth, reliability, fault tolerance, clock synchronisation, and membership services [2].

It has been argued that the best way to design a safe and reliable system is to make things more static [2][6]. By removing dynamic behaviour, the exact sequence of events can be determined in advance. This idea is exploited by time-triggered protocols in order to design dependable real-time systems [2]. However, the major drawbacks of static systems are the lack of flexibility in reacting to sporadic events and that they are not able to adapt to changes in the environment. Another disadvantage of these protocols is that they can only operate in a homogenous environment due to their strict communication requirements to ensure the safety and reliability of the system. This no longer matches the heterogeneity of today's complex systems.

The term flexibility refers to *configurational flexibility* throughout this thesis. The precise meaning of configurational flexibility will be elaborated in more detail in the upcoming chapters. The objective of the present research was to analyse the shortfalls of existing TTA based communication protocols, investigate how this can be overcome, and propose a comprehensive solution to this problem in the form of a protocol, its associated principles, and a proof-of-concept implementation. We will explore, how flexibility can be achieved despite the rigid timing requirements of safety-critical real-time systems. The impact of flexibility on safety services and how to handle that impact at the protocol level is also a contribution of this research. Furthermore, engineering real-time communication at a higher level using a subsumption architecture is another contribution of this thesis that demonstrates how the emergent complexity in designing a flexible protocol can be reduced.

1.3 Contribution of the Thesis

Flexibility and dependability are the two parameters that are often considered as contrary to each other and choosing between them to solve an engineering problem is a hard task [22][23]. There is a strong argument in the literature that to achieve and verify dependability, static prior knowledge about the sequence and timing of state changes is essential [2][6]. This idea has been exploited by the Time-Triggered Architecture (TTA) to produce more dependable real-time systems [6].

This thesis analyses how to overcome the fundamental shortfalls of the existing TTA based communication protocols. Two major incumbents are the Time Triggered Protocol and FlexRay. These protocols are widely used in safety-critical real-time systems. A number of other time-triggered protocols, developed on top of event-driven communication especially protocols under Time Sensitive Networking (TSN) domain were also explored in this thesis.

As an initial case study, a brake-by-wire case study is used to analyse the need for dynamic communication schedules in a TDMA round. I will show that functionality of each node connected to an Anti-Lock Braking (ABS) system is different and so are their payload requirements. The length of communication slot of each node is configured according to its payload requirements inside a TDMA round which significantly reduces gross overhead time. This work has been published in the proceedings of 20th IEEE Asia-Pacific conference on communication (APCC-2014)[Appendix A, Publication 3].

The span of communication schedules is expanded over a number of TDMA rounds in an improved iteration of the proposed approach. A number of use-cases such as Advanced Driver Assistance Systems (ADAS) in an autonomous vehicle, Unmanned Aerial Vehicles (UAVs), and healthcare systems based on Internet-of-Things (IoT) are investigated that exhibit this need for different transmission schedules during different TDMA rounds of a cluster cycle and in different modes of operation. The proposed approach improves channel utilisation nine times as compared to existing TTA-based communication protocols. This work has been published in an international journal of IEEE Access-2022 [Appendix A, Publication 1].

This thesis also contributes formal verification as a means to validate and ensure the correctness of the proposed approach. Protocol behaviour under different faults is analysed and verified by using a formal verification model. This work has been published in an international journal of IEEE Access-2022 [Appendix A, Publication 1].

This thesis further presents the implementation of the proposed approach utilising a high level subsumption architecture. The ability of Logic Labelled Finite State Machines (LLFSMs) is demonstrated to show that hierarchical module interactions can be utilised towards the implementation of testable, safety-critical, real-time communication protocols that transcend the inherent limitations of prior approaches. This work has been published in the proceedings of 11th International conference on Evaluation of Novel Software Approaches to Software Engineering (ENASE-2016)[Appendix A, Publication 2].

1.4 Thesis Outline

This thesis is divided into seven chapters. Following this introductory chapter, the rest of the thesis is organised as follows:

Chapter 2 explores to the basic concepts of real-time systems and real-time communication. The fundamental relevant material and definitions are presented in this chapter. Some key requirements such as timeliness, dependability and flexibility are discussed, which distinguish between real-time and non real-time systems. Two basic communication architectures, event-triggered and time-triggered architectures along with their characteristics are compared and analysed.

Chapter 3 discusses relevant communication protocols in the literature used for real-time communication in safety-critical real-time systems. An in-depth analysis of each protocol is presented in this chapter. The tradeoff between predictability and flexibility is discussed and an analysis shows how existing communication protocols are more tilted towards predictability and less towards flexibility. Protocols based on time-triggered architectures such as the Time-Triggered Protocol and FlexRay are analysed in more detail as well as communication protocols build on top of the event driven approach such as Time-Triggered Ethernet and other approaches under the domain of Time Sensitive Networking (TSN). A detailed analysis of the limitations of these existing protocols are presented in this chapter.

Chapter 4 addresses the issue of TDMA based slot length configuration of each node in a TDMA round of a cluster cycle. A brake-by-wire case study is used to analyse the impact of equal-length time slots over bandwidth and channel utilisation. We investigate what happen when each node in the system has different transmission payload requirements. An autonomous vehicle case study is presented that shows the need for flexible communication schedules.

Chapter 5 evaluates an implementation of the proposed approach. We investigate how complexity can be overcome by using a high-level subsumption architecture. The ability of LLFSMs is demonstrated to handle elaborate hierarchical module interactions that is utilised in the implementation of testable, safety-critical real-time communication protocols.

Chapter 6 analyses the complex task of preventing single points of failure. My contribution is a bus guardian analysis investigating how a faulty node can be blocked to transmit outside its assigned communication schedule even when we have a flexible communication schedule. Key fault scenarios are discussed and the role of bus guardians is analysed to tackle these faults. A verification model is presented to formally verify the timing parameters for transmitter, receiver, and bus guardian nodes.

Chapter 7 presents the conclusions from this thesis.

Appendix A presents the publications that have resulted from the work contained in this thesis.

Chapter 2

Real-Time Systems

This chapter presents a literature review in the context of an architectural view of real-time systems. It explains the difference between real-time systems and non real-time systems, including the fundamental concept of a *deadline*. The main focus of this chapter is on real-time communication, particularly the requirements of distributed real-time systems. To this end, we compare two key communication architectures, Event-Triggered (ET) and Time-Triggered (TT) systems.

2.1 Real-time Systems

A real-time system needs to process information and respond within a specified time frame [2]. In real-time systems, the correctness of system behaviour (sequence of outputs in time) not only depends on the logical computational results but also on the physical time when these results are produced [2]. A real-time system changes its state as a function of physical time [2].

Example: A chemical reaction continues to change its state even after its controlling computer system has stopped [2]. Therefore, a real-time system can be decomposed into a set of subsystems such as the controlled object, the real-time computer system and the human operator [2] as shown in Figure 2.1.

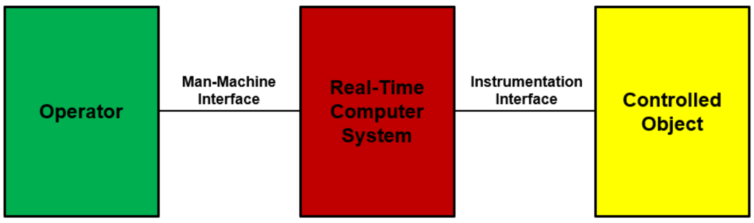

FIGURE 2.1: Real-Time Systems [2, p. 2]

A real-time computer system must react to stimuli from the controlled object or the operator within time intervals dictated by its environment [2]. The instant at which a result is produced is called a deadline [2]. A deadline is the time limit by which a task must be completed [2]. Efficiency of a real-time system is not measured based only on the logical results of computation but also depends on the timing of the results [24]. Deadlines can be classified into soft, firm and hard deadlines.

Soft deadline: If the result has utility even after the deadline has passed, the deadline is classified as soft deadline [11]. For example, a result is produced beyond the deadline, and the result is still effective and required by the system, then this is classified as a soft deadline [11].

Firm deadline: If the result has no utility after the deadline has passed, the deadline is classified as firm deadline [11].

Hard deadline: If a catastrophe could result if a firm deadline is missed, the deadline is considered as hard deadline [11]. In simple words, we can say that a deadline is considered a hard deadline if there are severe consequences of missing a firm deadline [2][25].

On the basis of soft and hard deadlines, a real-time system can be divided into two types i.e. a soft real-time system and a hard real-time system.

2.1.1 Soft Real-Time Systems

A soft real-time system is based on soft deadlines i.e. missing a deadline by a soft real-time system will have no catastrophic effect on the system environment [2]. Often in a soft real-time system, a performance degradation during a peak load situation can be tolerated.

Example: Consider the case of an airline reservation system. If the system is unable to keep up with the demands of the users, it just extends the response time and forces the users to slow down [2].

Example: In a factory automation system, a robot arm is collecting items from a conveyor belt must have a hard deadline to grab the correct item at the right time [19]. If the robot arm misses this deadline then the item will be out of its reach. Failure of the robot arm to grab the item from the belt within the specified deadline, may require production to be stopped, marking the system inefficient and incorrectly designed [19], but may not otherwise have catastrophic consequences.

2.1.2 Hard Real-Time Systems

A hard real-time system is based on hard deadlines i.e. missing a deadline by a hard real-time system may have severe consequences for the system or environment, potentially leading to catastrophic outcomes [2]. A hard real-time system can also be referred to as a safety-critical real-time system. Many research works [26][27][28] in the area of real-time systems focus on how it can be assured that these hard deadlines will be met. A real-time system that must meet at least one hard deadline is known as a hard real-time or a safety-critical real-time system [2].

Example: If brake-by-wire system does not respond within a few milliseconds when brakes are applied then this may result in catastrophic system failure [2].

Example: An accident could happen if a traffic signal at a road before a railway crossing does not change to red before the train arrives [2].

The research work presented in this thesis mainly focuses on hard real-time systems. A deadline is not the only criterion to define timing constraints, there are other factors that have equal importance in defining the behaviour of real-time systems, such as network latency, throughput and jitter to name a few [19]. Peak-load scenarios must be fully defined for hard real-time systems so that they can guarantee they meet the deadlines in all situations [19]. Active replication or redundancy mechanisms play a vital role for error detection in hard real-time systems [19]. By contrast, roll-back and recovery mechanisms are not useful for hard real-time systems because it would be hard to guarantee deadlines as roll-back and recovery mechanisms can take longer than required [19].

2.1.3 Distributed Real-Time Systems

In a real-time control system, the system interacts with real-world objects [3]. For instance, in a factory automation and control system, a variety of machines are widely distributed and interact with each other through a real-time communication network to complete the assigned task [3] as shown in Figure 2.2. Similarly, in an automotive setting such as brake-by-wire [29] and steer-by-wire systems [30], sensors are connected with computers all through the vehicle e.g., by means of bus technology.

As distributed control systems have to interact with real-world objects, these systems have timing constraints that must be supported by an underlying communication subsystem [2]. The communication subsystem is subject to timing constraints in the form of bounded network latency [20, 21]. This plays a major role in real-time communication. Besides the importance of timely delivery of messages, reliable message delivery is also critical [2].

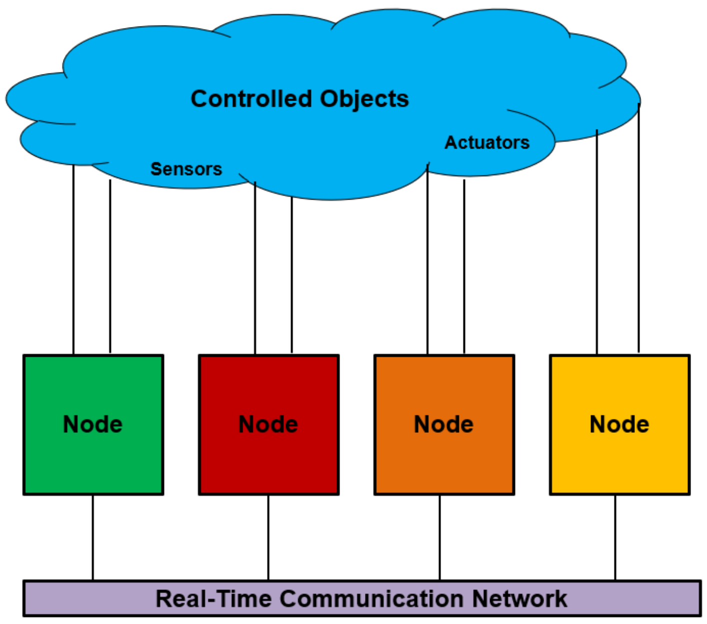

FIGURE 2.2: Distributed Real-Time Systems [3]

2.2 Real-time Communication

The previous section introduced the concept of *deadlines* to define real-time systems. In real-time communication, it is essential to assign deadlines to messages communicated over the network. Based on these deadlines, the timing behaviour of the system can be marked as correct or incorrect [2]. In distributed real-time systems, control is distributed over a number of nodes connected via a network. Some nodes rely on the information provided by other nodes as an output to perform their operations [2]. Therefore, distributed real-time systems heavily rely on their communication subsystems, which are responsible for timely and error-free delivery of the information among distributed nodes [2]. Consequently, for safety-critical real-time systems, timeliness, dependability, and flexibility are important factors in order to perform the behaviour required for successful operation [2].

2.2.1 Timeliness

The main difference between real-time and non-real-time communication is *Timeliness*. Timeliness in real-time communication can be achieved by ensuring short message-transport latency and minimal jitter [2]. To achieve short message-transport latency in a distributed real-time system, the time required to take the readings from a sensor node, and after computation, forwarding the result to an actuator node should be as small as possible [2]. Jitter is the difference between worst-case and best-case message transport latency [2]. Therefore, to ensure timeliness, this difference must be as small as possible, known as minimal jitter [2].

The underlying communication subsystem of any safety-critical distributed real-time system is responsible for the delivery of a message from a transmitting node to the receiver node within the defined time-line, ensuring that any corresponding deadline will be met [2].

2.2.2 Dependability

Dependability is a key characteristic of any safety-critical real-time system [2]. In many distributed systems, communication reliability is achieved by using a re-transmission mechanism in case of message loss [2]. For instance, in the Positive Acknowledgment and Retransmission (PAR) protocol, a sender must wait for an acknowledgment from the receiver after transmitting a message [2]. If no acknowledgment has been received after a specific time, the sender will retransmit the message and this process will continue until the message is transmitted successfully or a permanent failure is reported [2] as shown in Figure 2.3. TCP used this mechanism for reliable delivery of messages.

However, this mechanism is not suitable for safety-critical real-time systems, because it can add significant delays to the communication.

Fault-Error-Failure Chain:

Failures are usually recorded at the system boundary [31]. They are basically errors that have propagated to the system boundary and have become observable. Faults, Errors and Failures operate according to a mechanism [31]. This mechanism is sometimes known as the Fault-Error-Failure chain [31].

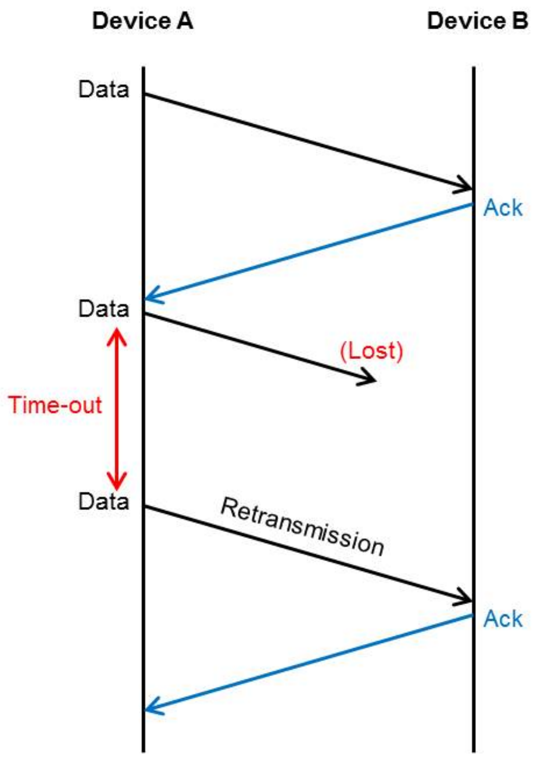

Figure 2.3: Positive Acknowledgment and Retransmission

Fault: A fault is a defect in a system [32]. This may or may not lead to a failure as a system may contain a fault, but its input and state conditions may never cause this fault to be executed so that an error occurs; and thus that particular fault never exhibits as a failure [32].

Error: An error is a discrepancy between the intended and actual behaviour of a system inside the system boundary [32]. When activation of a fault makes some part of the system to enter in an unexpected state, an error occurs [32].

Failure: A failure is an instance in time when a system displays behaviour that is contrary to its specification [32]. It can be prevented by using fault tolerance techniques so the overall operation of the system will conform to the specification [32].

In distributed real-time systems, communication reliability can be achieved through redundancy [2]. For example, robust channel encoding (forward error correction) or redundancy through transmitting replicated copies of a message on replicated communication channels can reduce the probability of message loss in the presence of a fault e.g., errors can be detected through a Cyclic Redundancy

Check (CRC) [33], where each message contains the CRC field that can be used at receiver side to check the integrity of the message [2].

Hardware component faults should be detected and reported by the communication subsystem to all other functioning components of the system so that these other components can handle the corresponding errors, avoiding a failure scenario [2].

Example: In a brake-by-wire system, if the brake system of one wheel has failed, then the other three wheels must be aware of this failure so that they can redistribute the brake force between them in a way that car can be stopped safely with only three braking wheels.

2.2.3 Flexibility

A real-time communication protocol must support flexibility to accommodate changes in the environment without any modification in the protocol [19]. For example, in an avionics system, the operation of system components is different during different operational phases, such as taxiing on ground, landing and during autopilot engagement [19]. Therefore, the underlying communication protocol must support the changes in the environment and must be able to react accordingly without significantly increasing the complexity of the protocol on the system [19]. This is a trade-off that is difficult to reconcile, and thus most of the existing safety-critical distributed real-time communication protocols lack flexibility, as we will explore in more detail in the next chapter.

2.3 Communication Architectures

This section discusses the two widely used communication architectures for real-time communication, Event-Triggered (ET) Communication and Time-Triggered

(TT) Communication. In light of the above mentioned communication requirements, this section will discuss their suitability for distributed safety-critical real-time systems.

2.3.1 Event-Triggered Communication

In Event-Triggered (ET) systems, all the system-related activities such as processing, communication, and the like, are based on the occurrence of significant events [12]. A significant event can be defined as a change of state in a real-time entity or object and this change of state is realised through an interrupt mechanism in ET systems [12][34]. The unpredictable nature of many events in ET systems requires a dynamic scheduler to handle these kind of events. As there is a possibility that multiple events can occur concurrently which may lead to significant delays in processing and communication [12].

In event-triggered communication, a sender node will transmit the message on the occurrence of an event [12]. This message will be placed in a send queue at sender side until underlying communication protocol is ready to transmit the message over the channel [12]. The readiness of communication protocol means giving the right to transmit to the sender node over the communication channel [13, 14]. A number of techniques exist in traditional, event-driven communication architectures to avoid collisions. A typical example is the Carrier Sense Multiple Access (CSMA) scheme that is based on collision detection (CSMA/CD) [13] or collision avoidance (CSMA/CA) [14]. In a collision avoidance system, each node has to sense traffic on the channel and wait until there is no traffic on the channel [14], while in a collision detection scheme, after detecting a collision all nodes have to back off (terminate their transmission) and retransmit after waiting for some random amount of time [13] as shown in Figure 2.4.

Due to their probabilistic nature, these techniques result in unpredictable channel access (and consequently, delivery) latency and are only suitable for non-safety-critical real-time systems, since best-case message delivery times can differ greatly

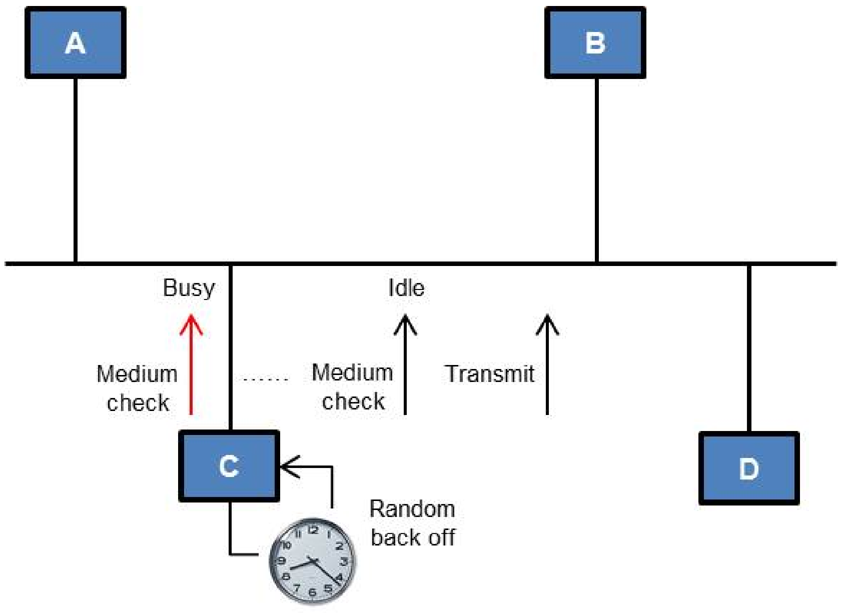

FIGURE 2.4: A Simple CSMA/CA based Scenario for Channel Access

from worst case scenarios and thus missing deadlines needs to be tolerable [2, 12, 34].

After the arrival of a message at the receiver the message is placed in the receive queue until accepted by the receiver after performing a series of tests to check whether the message is intact or not [2]. Involvement of queues to hold the message before transmission or acceptance has significant threat of queue overflow [2]. For example, the send queue can overflow if the transmission rate of the sender is larger than the network capacity and receiver queue can be overflow if the reception rate of the receiver is lower than the delivery rate of the network [2].

2.3.1.1 Characteristics of Event-Triggered Communications

Dynamic scheduling in ET systems make them more responsive than TT systems during low or average load conditions, as the processing of a task depends on its actual execution time, rather than maximum execution time as in TT systems [34]. The use of dynamic communication schedulers simplifies the design phase of ET systems compared to TT systems, where communication schedules are created and verified manually [34].

The ET based control paradigm is often preferred due to its higher flexibility and resource efficiency in soft real-time or non safety-critical applications [12]. ET architectures support dynamic resource allocation and resource sharing strategies [34]. The provision of resources in ET systems can be biased towards average demands, thus, during worst-case scenarios, timing failures are allowed to occur in favour of cost-effective solutions [34]. In ET systems, resources can be multiplexed among different applications while providing probabilistic guarantees for communication latencies only if the correlation between resource usages of different applications is known [35].

However, while ET systems offer the biggest amount of flexibility no analytical guarantees can be given for the performance of ET systems during peak system loads [12]. Communication protocols in ET systems are event-triggered. Therefore, only the sender knows the exact time when a message has to be transmitted [34]. Error detection in communication is based on a timeout parameter, with the sender waiting for a positive or negative acknowledgment from the receiver [12]. Transmission of explicit acknowledgment adds time in the case of failure and thus, can add significant and unpredictable delays in communication [12]. Therefore, assessing the worst-case execution time, and thus guaranteeing to meet deadlines becomes complex and quickly infeasible, making this approach unsuitable for most safety-critical real-time systems [12, 34].

For example, consider a TCP connection that is used to ensure reliable message delivery by using the PAR mechanism [12]. As TCP is normally expected to deliver messages with tight constraints, errors can introduce significant delays in communication due to timeouts and retransmission [12]. As TCP and, in general, ET communication are difficult to predict in the presence of faults, they are unsuitable to use for responsive or dependable safety-critical real-time systems [12].

Moreover, there is the possibility that multiple events can occur at the same time, e.g. as a result of communication failure [34]. This is referred to as *Event showers* that can cause extreme delays in processing events [34].

2.3.2 Time-Triggered Communication

Time-triggered communication is based on TDMA (Time Division Multiple Access), where the right to transmit over a shared communication channel is based on pre-defined time-slots [12]. A prominent example of a TDMA-based medium access is the Time-Triggered Protocol (TTP) [33], which is discussed in detail in Section 3.4.

In order to avoid a collision in TDMA based medium access control, each node is allocated a different time-slot and it can only transmit over the communication channel within this slot [12]. Slot allocation is done offline and all the nodes have the information about their allocated slots a *priori* as shown in Figure 2.5.

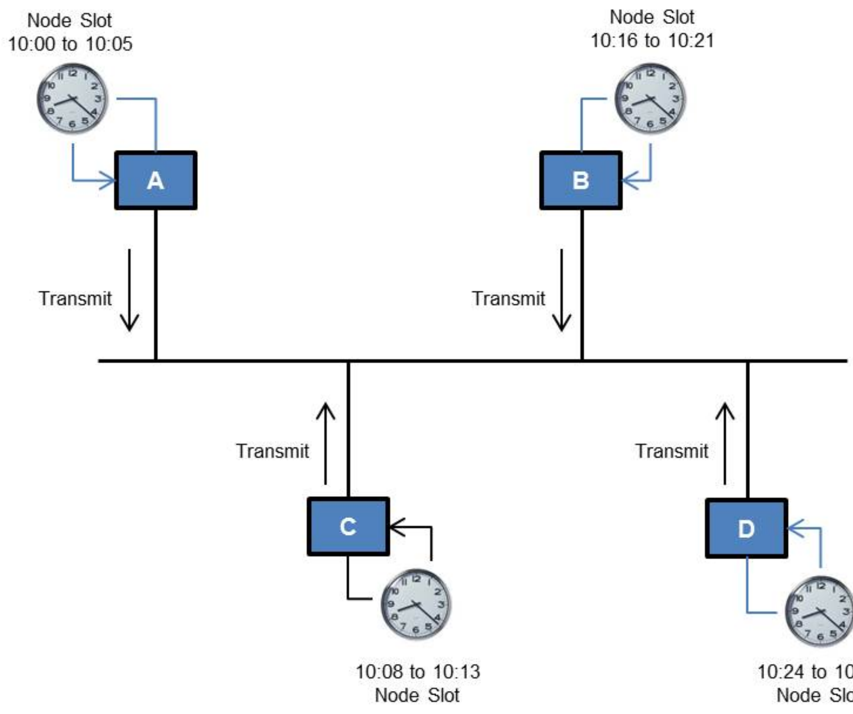

FIGURE 2.5: A Simple TDMA based Scenario for Channel Access

As all the communication scheduling is determined in advance, it is easy to find the worst-case response time [12]. The key benefit of deterministic communication is to guarantee hard deadlines for safety-critical real-time systems [19].

Each node in a distributed Time-Triggered (TT) system contains only one interrupt, the periodic real-time clock interrupt [2]. In a TT system, a processing or communication activity is initiated at a predetermined tick of a clock [2]. A global time is formed by synchronising clocks of all the nodes in a distributed TT system [2]. This global time is used to time-stamp the observation of the controlled object [2]. The granularity of the global time must be chosen such that the temporal order of any two observations made anywhere in a distributed TT system can be established from their timestamps with adequate accuracy [2].

2.3.2.1 Characteristics of Time-Triggered Communications

The predictable nature of time-triggered communication has made TT systems capable of handling peak load scenarios without performance degradation [12]. In TT systems, receivers know exactly when they will receive a message from the sender, due to its pre-defined communication schedule [12][2]. If a message is not received by a receiver within the predefined time-frame, this will be considered a loss of communication [2]. Therefore, TT systems do not require any explicit acknowledgment or timeout mechanism [2, 34, 36]. The temporal behavior of TT systems remains the same, as long as maximum execution time of a task stays within its deadline [34]. Consequently, TT systems are much more reliable than ET systems and their reliability can be predicted analytically, which is an important aspect for any real-time system [34].

Alterations in TT systems require redesigning all communication schedules. Distributed TT systems must deal with the requirement of clock synchronisation to function correctly [6]. Therefore, all the distributed components of a TT system must maintain a global time base and their clocks must be synchronised with each other [6][36]. Responsiveness of TT systems is lower than that of comparable ET systems during low or normal load situations, as they need to wait for the worst possible execution time of a task to have elapsed [34]. Designing a communication schedule before the system deployment is a complex part of TT systems, because anything related to timing that is not completely known in advance cannot be handled efficiently [34].

2.3.3 Discussion

In general, it depends on the application whether an event-triggered or a time-triggered system behaviour is suitable, as both approaches have specific characteristics making them suitable for specific requirements. Considering the requirements of safety-critical real-time systems, i.e. timeliness, dependability, and safety, it can be argued that a time-triggered approach is more suitable for these kind of systems because of its deterministic communication. Unlike an ET approach, TT protocols provide a mechanism for detecting communication errors without the need of explicit acknowledgment. Therefore a time-triggered approach reduces the significant difference in timing between the best and worst case in communication and hence, guarantees the requirements for worst-case responsiveness of safety-critical real-time systems.

Chapter 3

Real-Time Communication Protocols

This chapter gives an analysis of communication protocols used in safety-critical real-time systems. This analysis presents the trade off between predictability and flexibility of communication protocols. The discussion will start in a chronological order on the basis of the evolution of these protocols. The shortcomings of the existing protocols for safety-critical real-time systems will be analysed and discussed in this chapter. Although the wireless communication is not in the scope of this thesis but there will be an insight of why wireless architectures are not suitable for safety-critical real-time communication and what the possible areas are where an integration of wired and wireless architecture can be used.

3.1 Controller Area Network (CAN)

In the mid-eighties, a serial data communication bus known as Controller Area Network (CAN) [37, 38] was developed by Robert Bosch for the German car industry. Since then, CAN has been adopted by the automotive industry worldwide for data communication. The major reasons that drove the development of CAN were provision of real-time communication, combined with reduced cabling size and weight [37, 38]. Networking protocols in industrial devices were actively adopting CAN, prominent examples include DeviceNet [39] and CANOpen [40]. Examples of CAN devices include engine controllers (ECUs), transmissions, Anti-lock braking systems, lights, power windows, power steering, and the like.

This section briefly describes the communication mechanism for all nodes connected through the CAN bus. A detailed description of the protocol can be found in [37, 38]. CAN consists of three layers with reference to the OSI (Open System Interconnection) model [41][42]. These layers are the physical layer, data link layer, and an optional application layer. The data link layer is specified by [43] while physical layer specifications are discussed in ISO-11898 [44]. At the application layer, high level protocols are used such as CAL/CANOpen [45] and CAN Kingdom [46].

CAN is deployed in a bus topology and belongs to the class of event-triggered protocols, where communication activities are initiated on the occurrence of significant events [37]. The basic channel access mechanism is based on CSMA/CA, where a node can only transmit its message over the communication channel after sensing that the channel is idle [14]. If multiple nodes intend to transmit messages at the same time, then the node with the highest priority message will transmit first [37].

To achieve this, the CAN bus behaves as a logical-AND electrical network when monitored in a dominant-bit format [37]. The dominant bit is interpreted as zero and a recessive bit is interpreted as one [37]. If all nodes are transmitting recessive bits then receivers perceive the bus as carrying a recessive bit [37]. If one or more nodes are transmitting dominant bits, then all nodes will see a dominant bit on the bus [37].

The priority of a message is identified through an 11-bit arbitration field [37]. When two or more nodes are trying to transmit data, they monitor the bus for inactivity [37]. When the bus is idle, all nodes waiting to transmit will send a Start of Frame (SOF) bit, potentially simultaneously [37]. SOF is a dominant bit, so all nodes will see that dominant bit on the bus [37]. After this, each

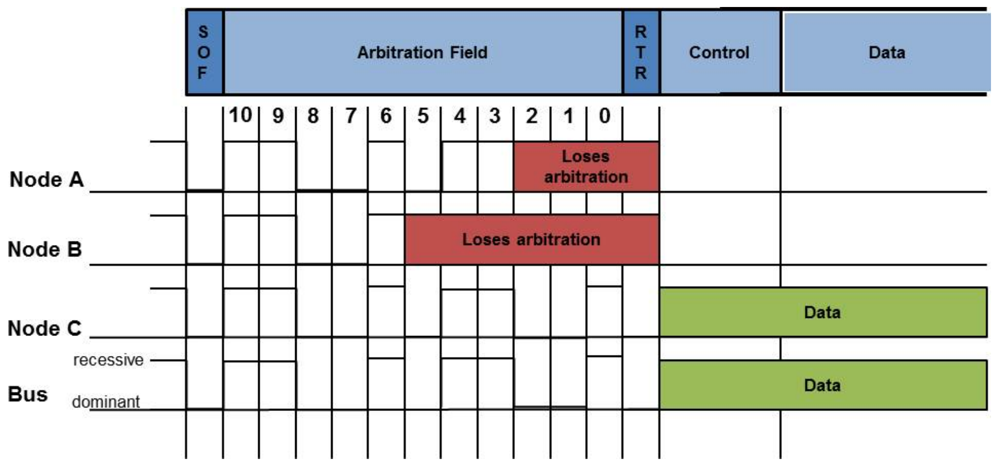

Figure 3.1: Arbitration Mechanism in CAN

node will transmit the arbitration field on the bus [37]. The arbitration field has an 11-bit binary value with the most significant bit being transmitted first [37]. During the arbitration process, if a node transmits a recessive bit, but detects a dominant bit on the bus, this means there is another node with a higher priority message competing for the arbitration [37]. Then the node with the lower priority message stops transmitting and allows the node(s) with high priority messages to continue [37]. This continues bit by bit, so at the end of arbitration only the node with the highest priority will be left, transmitting its data on the bus [37]. This higher priority node will continue to transmit until its data is completely sent or an error is detected [37]. Nodes with lower priority messages will wait until the next idle interval on the bus [37]. Then they will try again to send their messages, after winning the arbitration process in the same way as discussed above [37]. This whole arbitration mechanism is shown in Figure 3.1.

In CAN, if a node starts transmitting high priority messages continuously, then low priority message from other nodes will not get a chance to transmit. The indefinite amount of time wait for these low priority messages can create a starvation scenario. Consequently, if the transmission load on the communication channel increases, the worst case transmission time of messages varies significantly and cannot be predicted or bounded. For guaranteed temporal deadlines of safety-critical real-time systems, message delivery must occur within a given deadline [2].

Another issue with CAN is propagation delay. For example, one node can see a recessive bit on the channel while another node can see a dominant bit on the channel due to different propagation delay. Due to the medium access technique (priority-based Carrier Sense Multiple Access/Collision Resolution), the maximum data rate that can be achieved with CAN, essentially depends on the bus' length [47]. For example, the maximum data rate for 30 and 500 meter buses are respectively 1 Mbit/s and 100 kbit/s [47].

CAN is basically an event-triggered communication protocol with a flexible communication schedule. A node can transmit any time at the occurrence of an event and therefore has an efficient response time during low or average load conditions. But this flexibility comes at the cost of reliability. Any node can monopolise the communication network because of the arbitration mechanism in CAN. It is hard to guarantee the deadlines due to starvation issues in CAN. This is particularly precarious in the presence of faults, as single point of failure cannot be ruled out.

3.2 ARINC-629

ARINC-629 [48] is a communication protocol for a bidirectional and multiple access data bus and is capable of transmitting safety-critical and non-safety-critical information. It is designed for modular avionics systems [49] and more specifically used in Boeing 777 [50].

The CSMA/CA protocol, which allows any bus node to start transmitting after sensing the bus being idle, avoids possible collisions on the communication channel [14]. But still, there is a room for collisions due to propagation delays on the transmission line. For example, a node waiting to transmit may see an idle bus while another node has already started transmitting. To cope with this situation, the CSMA/CD protocol supports collision detection so that nodes can determine whether their attempt to transmit was successful or not [13]. In case of collision detection, nodes stop transmitting, and after waiting for a random delay, they attempt to transmit again, provided the bus is idle at that moment [13]. A random back-off mechanism is used to avoid permanent collisions, but this may result in poor channel utilisation and severe throughput problems during peak-load scenarios [13]. This mechanism also introduced network monopolisation issues [48]. If a node starts transmitting after sensing that channel is idle, then there is no mechanism to avoid network monopolisation by that node, because it can utilise the channel for transmission as long as it wants [48]. This will prevent other nodes from transmitting over the communication channel [48].

ARINC-629 was designed to handle the issues of CSMA/CA and CSMA/CD protocols [48]. It is a time-driven protocol but with a capacity to transmit non-periodic data after transmitting periodic data [48]. Other protocols that transmit periodic and non-periodic data in a similar way include WorldFiP [51] and ProfiBUS [52]. WorldFiP is using master node to control the medium access [51] while ProfiBUS is using token passing mechanism to transmit the data on the channel [52].

The channel access in ARINC-629 is based on three timers (timeout parameters), a Synchronisation Gap (SG) timer, a Terminal Gap (TG) timer, and a Transmit Interval (TI) timer [48]. First, the set of nodes that want to transmit is admitted to a waiting room [48]. All the nodes in the waiting room can transmit their messages before a new node is allowed to enter in the waiting room [48]. The value of the SG timer is the same for all the nodes and it is used to control the entrance of nodes into waiting room [48]. The TG timer has a different value for each node and it is therefore known as a "personality" timer [48]. TG is used to control access of a node to the communication channel [48]. The TI timer is used to prevent a node from monopolising the channel and it is identical for all the nodes [48]. The relationship between these timers can be defined as:

$$SG > Max(TG_i) \tag{3.1}$$

$$TI > SG \tag{3.2}$$

The detailed operation of the protocol regarding channel access is best explained by taking an example of two nodes P and Q, that want to transmit over the communication channel [4] as shown in Figure 3.2. Let us assume that TG_P of node P is shorter than TG_Q of node Q [4]. Initially both nodes will wait for a period of silence in the communication channel that is longer than SG [4]. Then both nodes will enter in the waiting room where they have to wait for another period of silence corresponding to their individual TG timers i.e. TG_P and TG_Q [4]. As TG_P is shorter than TG_Q then node P will start transmitting if the bus is idle when its TG_P has elapsed [4]. At the start of transmission, node P will set its TI to block any further transmission activity before the transmission activity of all other nodes admitted in the waiting room is completed [4]. This mechanism makes it impossible for a single node to monopolise the communication network [4].

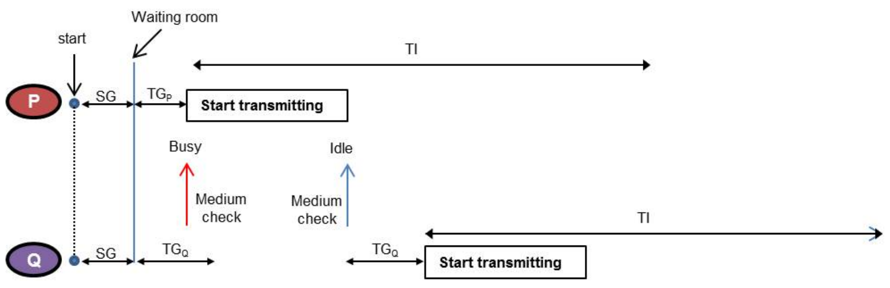

FIGURE 3.2: Transmit Logic of ARINC-629 [4]

As soon as node P has started its transmission, node Q should back off until P has finished its transmission [4]. After P has finished, Q needs to wait for TG_Q again and it will start its transmission if there is no bus activity on the communication channel at the point of TG_Q timeout [4].

ARINC-629 [48] is a communication protocol designed specifically for avionics systems [53]. The Boeing 777, for example, is using ARINC-629 for control and many related safety-critical functions [50]. ARINC-629 utilises a data bus that is bidirectional and allows access for transmitting both safety-critical information (SCI) and non-safety-critical information (NSCI) [48] [4]. Monopolisation of the communication network is handled in ARINC-629 by controlling network access by using three different timers as discussed above. Channel access in ARINC-629 is loosely based on time slots regulated by a transmission gap (TG) timer [48].

The protocol uses a collision avoidance approach (CSMA/CA) [14] for bus arbitration [48]. The value of the TG is different for each node in order to prevent simultaneous channel access by different nodes [48]. A node starts listening on the communication channel and once its TG has elapsed, it starts transmitting its messages only if the channel is idle [48]. A node cannot transmit if its TG has elapsed but there is traffic on the channel. In this case, the whole procedure of the timer restarts afresh [48]. ARINC-629 is similar to other approaches [52][54] that transmit periodic and sporadic information. However, a timing analysis of ARINC-629 shows that the protocol supports periodic and sporadic traffic with deadlines, provided that the worst case sporadic traffic in the system is known [55]. In ARINC-629, a time slot is allocated to each node and it listens to the communication channel for inactivity [55]. If the bus is inactive during the allocated time slot, the node may begin to transmit [55]. But if the communication channel is busy, access is not attempted until the next time slot [55]. This slot allocation sets the time limit on the node for transmission so that it cannot monopolise the communication channel [55].

Importantly, in ARINC-629, nodes do not have a common knowledge about the communication schedule, so different nodes can attempt to transmit at any time. Due to this *unpredictability*, a global view and associated guarantees of system real-time behaviour cannot be determined a *priori*. The arbitration mechanism in ARINC-629 to avoid collisions on the shared channel makes it near impossible to predict worst-case response times.

ARINC-629 does not provide any significant support at the protocol level against faults introduced on the network due to interference. The protocol leaves all concerns regarding fault tolerance to the application. This means fault tolerance must be handled at application level. Furthermore, the channel access mechanism in ARINC-629 is not efficient and does not prevent starvation.

The inability of ARINC-629 to prevent starvation can be demonstrated with a simple example. If a single node (often termed a *babbling idiot*) continually occupies the channel, then other nodes will keep waiting indefinitely, as they cannot detect the channel as idle. Furthermore, fault tolerance is not handled at protocol level and it is assumed that all such concerns can be handled at application level, which is also not possible in this scenario.

3.3 Time-Triggered Model

The Time-Triggered Architecture (TTA) [6] is a composable architecture for the design of large real-time systems in the safety-critical domain. The main characteristics of TTA are common a notion of time in all subsystems as well as the provision of interfaces fully specified in both the value and time domains, known as temporal firewalls [2][6]. A large real-time system can be decomposed into subsystems such as a controlled object, an operator and a computational subsystem [2]. The interface between a controlled object and the real-time computer systems is called the instrumentation interface, which consists of sensors and actuators [2]. While the interface between an operator and the real-time computer system is known as the man-machine interface, consisting of input devices such as a keyboard [2]. A real-time system can change its state as a function of physical time [2]. The current state of a controlled object can be described by recording the values of its state variables at a specific instant [2]. It is not always necessary to record all the state variables, but a few are significant for a given purpose [56]. These significant state variables constitute a Real-Time (RT) entity [56]. The current picture of an RT entity is known as an RT image [2]. In other words, we can say that the observation of an RT entity in a controlled object is stored as a real-time image [2]. At any given point in time, an RT image is considered valid if it is an accurate representation of its RT entity in both the time and value domains [2]. However, an RT image is time dependent and thus it can be invalid with the progression of time [2]. The temporal accuracy of an RT image is the time interval during which it is considered temporally valid [57]. A TTA node can be considered as an RT object that provides the current RT image of its corresponding RT entity [58]. All elements of the TTA have access to a global time with known precision and must be fault-tolerant to avoid catastrophic consequences [59]. The progression of global time is used as a control signal to transmit and receive messages and for monitoring the temporal accuracy of the messages [59]. An interface has a huge importance in the Time-Triggered (TT) model [2]. It consists of a memory element between two subsystems and holds the RT images of the relevant RT entities [2]. The characteristics of interfaces in the TT model are described through temporal firewalls [2][6]. The temporal firewalls can be categorised as phase-sensitive or phase-insensitive [2]. A phase-insensitive temporal firewall is a data sharing interface that can be accessed by a subsystem at a *priori* known instant [2]. The information contained in the temporal firewall is considered temporally accurate for at least the defined temporal accuracy interval [2]. By contrast, on a phase-sensitive temporal firewall, the information remains accurate for a given, but much shorter temporal accuracy interval in the future at the instant when the information is delivered to the temporal firewall by a producer process [2]. Application tasks receiving phase-sensitive data elements must be synchronised with the sending task [2][60][61]. Otherwise, a state estimation task must be executed at the receiver [61]. Therefore, phase-insensitive data elements are preferred as they make the system more resilient and less tightly coupled [5][60][61].

For example, if a user process uses a data element such as a sensor value from a phase-insensitive firewall within a defined temporal accuracy interval after reading the value then the result will be temporally valid at the time of use[5]. Therefore, the data elements of the phase-insensitive firewall can be accessed at arbitrary instants without the risk that the RT image becomes invalid before it is used [5]. An example of information validity in a phase-insensitive firewall [5] is shown in Figure 3.3. On the other hand, a user process that uses a data element from a phase-sensitive firewall [5] must ensure that the time interval between its use of the information and the point in time of information delivery by the producer is less than the temporal accuracy interval [5]. An example of information validity in the phase-sensitive firewall is shown in Figure 3.4.

In the TTA it is assumed that sensors are intelligent [5]. This means that every sensor and actuator has a microcontroller with processing capability [5]. These

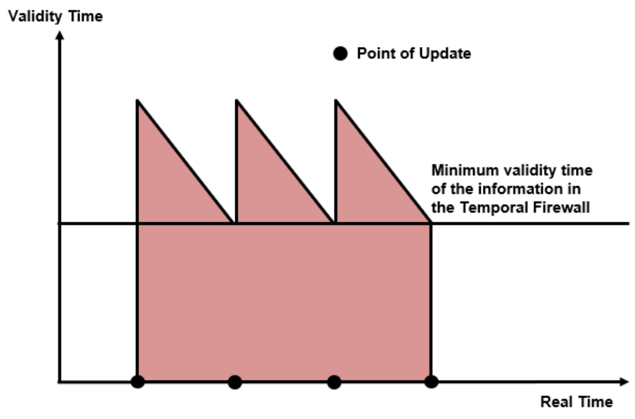

FIGURE 3.3: Information Validity in the Phase-Insensitive Firewall [5]

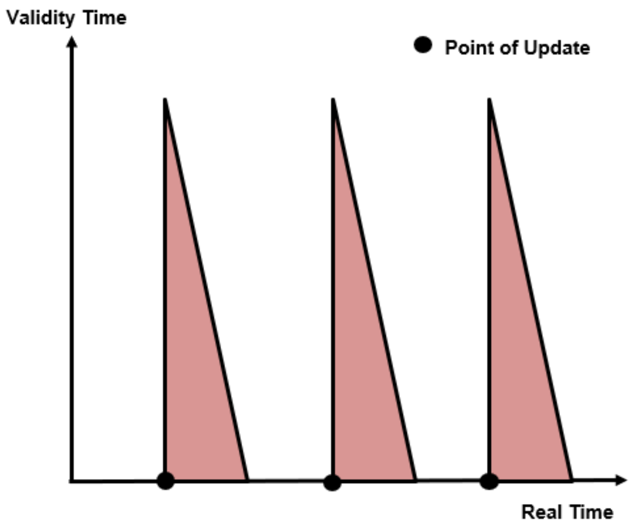

FIGURE 3.4: Information Validity in the Phase-Sensitive Firewall [5]

microcontrollers observe the associated RT entity and, after after converting the signal to a specific format, they deliver it to a TTA node in that format [60]. On a serial bus, only a single message can be transported at a time [60]. Therefore, some observation messages have to wait longer than others [60][5]. However, in the TTA the time difference between an observation of an RT entity and the delivery of corresponding RT images at the receiver is known a *priori* [5][60]. This time difference can be used by the intelligent sensors for state estimation [5][60]. Since each TTA node knows in advance about its channel access interval, sensors can use their knowledge of their personal time slots for this state estimation [5, 60]. At the receiving TTA node, the behaviour of the system will be as if all sensors values were observed at the same point in time. Thus, the temporal properties of a sensor subsystem are hidden behind the temporal firewall of a controlled object [5, 60, 62].

3.4 The Time-Triggered Protocol

The origin of TTP [33] can be found in the MARS (Maintainable Architecture for Real-Time Systems) project [63] that was started in 1979 and funded by the European Commission. The aim of this project was to build some baseline for an architecture so that within the next twenty years it became possible to implement a node for a distributed real-time system on a single inexpensive chip [64]. The mechanism of fail-silent behaviour of a node on the detection of faults was a key concept of MARS project [63] that was adopted by subsequent real-time communication protocols such as TTP [33]. Architectural components akin to MARS-like nodes, node clusters, global time base, fault-tolerant units and the like are still actively being used in TTP [33] and in other real-time communication protocols such as FlexRay [7].

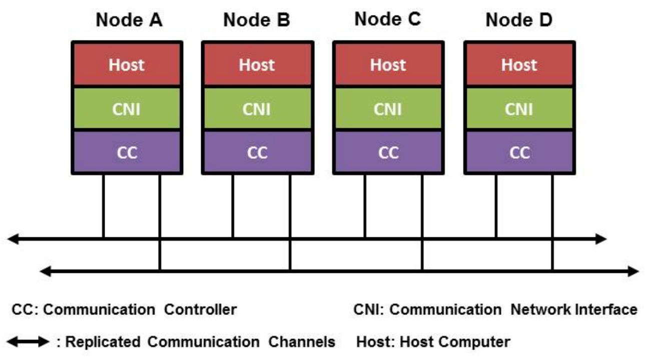
Node A
Node B
Node C
Node D
Host
CNI
CC
CC: Communication Controller
CNI: Communication Network Interface
: Replicated Communication Channels
Host: Host Computer

FIGURE 3.5: Structure of TTP cluster [6]

The development of TTP was led by Prof. Hermann Kopetz at the Technical University of Vienna [2] and is now being maintained by the TTA-Group since 2001.

TTP/C [8] is a variant of TTP [33] specifically designed to meet the requirements of SAE (Society of Automotive Engineers) Class-C standards [65]. Class C is a protocol class that includes safety-related features such as prevention against babbling idiot failure, omission failure, crash failure, etc [65]. Class C also requires low and bounded network latency as well as the provision of fault-tolerant, distributed clock synchronisation mechanisms [18]. The time-triggered nature of TTP establishes the ground for it to qualify for SAE Class C requirements [18]. TTP/C is widely used in safety-critical real-time systems such as avionics and automotive systems such as fly-by-wire and drive-by-wire systems [66].

3.4.1 TTP-Internal Operation

TTP implements its communication mechanism through a TDMA scheme with statically configured timing [33]. All nodes participating in the system are allowed to periodically access the channel on their turn for a fixed length time slot [33]. The duration is the same for every node's time slot [33]. The main communication component of TTP is a shared communication bus that interconnects distributed nodes with each other [33]. Each node consists of a host layer, a Communication Network Interface (CNI) layer and a Communication Controller layer [33] as shown in Figure 3.5.

3.4.1.1 Node start-up and reintegration

After starting up all the nodes operating in the system, each node will wait for its start-up time-out before accessing the communication channel [8]. Please note that each node has a different start-up time-out value and the node with shortest time-out will start the communication by sending an initialisation frame (I-frame) over the channel [8]. The I-frame is used to establish a global time-base by synchronising the clocks of all distributed nodes [8]. Collisions on the channel are possible [67] during the start-up phase of TTP if the start-up timer of two or more than two nodes elapse at the same time and they start transmitting I-frames at the same time over the communication channel [8]. To make sure no node is integrated on any of the colliding frames in case of a startup collision between two cold starters, a big bang mechanism is used [8]. In case of a startup collision between two cold starting nodes, the big bang ensures that no node will integrate on any of the collided frames [8]. However, if a collision is consistently detected by all nodes then big bang is not required [8]. It is possible that subsets of nodes may integrate on different cold starters and hence, create multiple cliques [8]. Therefore, to avoid such a scenario, the first received, correct cold start frame is rejected by all nodes and the listen timeout is restarted [8].

There are two types of frames in TTP, initialisation frames (I-frames) and normal frames (N-frames) [8]. The I-frames are used to form the global time base by synchronising the clocks of all the nodes during system start-up phase [8]. They are also used during normal operation (data transmission) at predefined intervals of time for reintegrating lost or failed nodes [8]. N-frames are used during normal operation to transport data [8].

3.4.1.2 Clock Synchronisation

Clock synchronisation [68] is a core requirement of time-triggered communication protocols [6]. In TTP, the nodes have physical clocks (internal or local clocks) and a predetermined schedule defined in their own copy of the MEDL [8]. Each node operating in the system times its slots (transmit or receive actions) in accordance with a predetermined schedule that is triggered by the progression of time [33]. For these node slots to be synchronised, the nodes' clocks have to be synchronised [68]. Any clock skew between sender and receiver is identified at the receiver side by using the difference between the expected arrival time defined in the MEDL and the actual arrival time of a frame [68]. The slots used for clock synchronisation are denoted in the MEDL (e.g., transmissions from nodes with cheaper, less accurate clocks may be disregarded) [8]. These slots are called resynchronisation points [8]. The Fault Tolerant Average (FTA) [59] is then used to correct the local clock based on the measured deviations. There are two phases to perform clock synchronisation. In the first phase, the difference is stored in a sorted array with a size of four where the highest and lowest value are discarded and then the average of the remaining two values is calculated [59]. This average value is used as the correction term to adjust the local clock's time [59].

3.4.1.3 Membership Service

The membership service of TTP informs all connected nodes about their status operating under the same cluster i.e. whether they are functioning correctly or not [8, 68]. This service makes TTP capable of determining any violation of the fault-hypothesis, so that a never-give up (NGU) strategy can be invoked immediately [8]. When there are not enough resources available to provide the minimum required service, then NGU is initiated by the TTP in combination with the application [8]. The NGU strategy is highly application specific e.g., if the cause of the outage is a massive transient fault, then the NGU strategy in some applications may freeze the actuators in their current state until a successful restart of the whole cluster has been completed [62] [8].

The membership is encoded in a membership vector whose size is equal to the number of nodes operating in a cluster [8]. Each node is assigned to a specified bit position of the membership vector and this bit is set to TRUE if node is operating correctly otherwise this bit is set to FALSE [8]. The membership point of a node is the periodic send instant of a message by this node [8]. The Controller-state (C-state) of a communication controller of a node consists of the current time and membership vector [8]. A 24-bit CRC is calculated over the message contents in concatenation with the C-sate of the sender to enforce the agreement on the C-state of all the nodes of a cluster [8]. At the receiver side, again a 24-bit CRC is calculated on the contents of received message in concatenation with the C-state of the receiver [8]. Any disagreement on C-states of sender and receiver or the transmission of faulty/corrupted message is detected through the negative result of the CRC check at the receiver side [8]. The message is discarded in either case and the receiving node assumes that the sender has been faulty [8]. This way, a correct node cannot be killed by a faulty node in the system [8]. In the above scenario, if the sender node has been correct then all other nodes have received the message correctly, which means only the receiving node must have been faulty [8]. Therefore, on its turn it will send a message with a wrong membership vector and all other working nodes will remove this node from their membership [8]. By contrast, if the receiving node is right, then the sender node will be considered faulty and will be removed from the membership [8].

It is important to note that, by using a membership service, TTP removes the need for any additional acknowledgment overhead or timeout requirements, as this can be achieved by checking the membership list of the successive sender [8].

3.4.1.4 Message Descriptor List-MEDL

The Message Descriptor List (MEDL) [8] is a data structure that contains all control information which is required during the initialisation phase and is stored in the communication controller's memory [8]. This makes the communication controller able to work independently, without requiring any control signal from the host after the initialisation phase [8]. The MEDL contains information such as:

- The address of CNI from where data can be retrieved and the points of time when this data could be sent [8] [69].
- The point of time when data should be received and the address of the CNI where received data can be stored. It also includes other additional information for the protocol [8].

3.4.1.5 Communication Network Interface-CNI

Importantly in TTP, there is a CNI layer between the host and communication controller [69]. The CNI is visible to the application software running on the host computer [69]. The CNI has a status area and a message area [69]. The status area is used for communication between the communication controller and the host, whereas the message area is used for the exchange of data sent and received by the node [69] [8].

3.4.2 Reliability and Fault-Tolerance in TTP

The TTP provides a high degree of fault-tolerance against a number of failures [68]. Fault-tolerant elements of TTP include:

- **Fail-Silence:** TTP nodes are designed in such a way so that they can detect faults in their own operation [8]. If a node detects a fault in its operation, it isolates itself from other correct nodes and will not participate in the system operations until it restarts itself in a healthy state [69] [8].
- **Bus Guardian:** Bus guardians are implemented at the hardware level in TTP [69] [8]. Each normal node has a bus guardian node in the bus topology of the TTP network where the bus guardian is synchronised with the cluster and knows the transmission schedule of the node [8]. A normal node that tries to transmit outside its allocated time slot is known as babbling idiot node [8]. The bus guardian physically prevents the normal node from transmitting over the communication channel outside its allocated time slot [8].
- **Replication of Components:** TTP provides the basis for fault tolerance through node replication and ensures replica determinism [8]. Use of duplicated communication channels makes it fault-tolerant in case of a single channel failure [68].

The predictable nature of the Time Triggered Architecture (TTA) [6], by comparison, provides a solid base to implement reliable and fault-tolerant distributed real-time systems [68] [33]. The Time-Triggered Protocol (TTP/C) [33][8] implements TTA communication by using a fixed, TDMA based channel access scheme, where all nodes are allocated static and equal length time slots [8]. The scheduling of time slots is done offline and all nodes know the exact time of transmission and reception of data [8]. This predictable nature of communication made it possible for TTP to provide effective error detection mechanism that ensures the maximum degree of fault tolerance, safety and availability. It is therefore largely deployed in dependable real-time systems such as automotive, aerospace, banking and the like.

However, on one hand this predictability supports reliable and timely delivery of messages and on other hand it makes TTP inflexible.

A node can have a different transmission payload during different operational modes. Therefore, the length required for its allocated time slot can vary during different modes. However, TTP does not support any dynamic transmission schedule that can allocate different time length slots to the same node during different mode of operation, that depends upon the transmission payload of that node.

A more flexible approach is introduced by using slot-skipping (TDMA/SS) [70] mechanism to improve channel utilisation. The basic concept of TDMA/SS is to skip the transmission slot of a node if it does not start sending within a predefined time in its slot [70]. The next node is permitted to send data before the scheduled time [70]. Channel utilisation can be improved by using this [70] and other similar approaches later [71]. However, these approaches are unsuitable for fault-tolerant, SCRT systems, as its flexibility compromises the determinism inherent in the distributed agreement achieved by the static schedule and fault-tolerant clock synchronisation, which is a basic requirement of fault-tolerance in the TTA.

3.5 FlexRay

FlexRay [7] is an automotive network communication protocol developed by the FlexRay Consortium [72] in 2000. It can be viewed as the combination of TTP [33][8] from the Technical University of Vienna and ByteFlight [73] from BMW. Byteflight was designed for safety systems where a short response time is required such as Airbag release application in automotives [73]. As these kind of systems have a very short mission time (in milliseconds) the probability of fault occurrence is very low, hence the Byteflight protocol forgoes require any fault-tolerance mechanism [74]. Without fault-tolerance support, Byteflight cannot be used for other control systems such as X-by-Wire systems which have a long mission time [74].

The structure of a node in FlexRay is the same as in TTP (See Figure 3.5). A node has three components, a host computer, a CNI and a communication controller [7]. The basic communication channel access mechanism is the same as in TTP. However, FlexRay adds flexibility by dividing the communication cycle into different segments [7, 75]. A communication cycle in FlexRay consists of a static segment, a dynamic segment and two other protocol segments called Symbolic Window (SW) and Network Idle Time (NIT) [7] as shown in Figure 3.6.

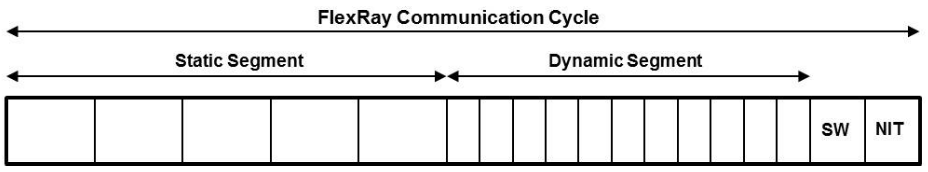

FIGURE 3.6: FlexRay Communication Cycle [7]

In the static segment, all nodes are allocated static and equal-length time slots to transmit safety-critical information, while the dynamic segment is based on dynamic time slots also known as mini slots (inherited from the ByteFlight protocol [73]) to transmit non-safety-critical information [7]. Channel bandwidth in the non-safety-critical part is shared on-demand among the nodes to ensure better bandwidth utilisation [7].

The SW is a fixed-length time slot used to transmit special symbols over the network to perform network management [7]. During the NIT in a communication cycle there is complete silence on the communication channel [7]. Communication controllers utilise this time to execute the clock synchronisation algorithm [7]. FlexRay utilises similar fault-tolerance mechanisms to TTP for the safety-critical part of communication. Both protocols are designed to address the same set of requirements for automotive systems, but there is a clear distinguishing line between their goals. TTP is tilted more towards safety while FlexRay towards flexibility.

FlexRay adds its flexibility only by providing support for non-safety-critical information, i.e. by introducing a dynamic segment for this information in each TDMA round [7]. However, the static segment of FlexRay that supports safety-critical communication uses the same approach with regards to channel access as TTP, i.e. the use of static and equal length time slots for all the nodes [7].

As FlexRay is also based on the TTA [6], reliable and timely delivery of messages is ensured through predictable real-time communication where communication schedules are predefined and all the nodes of a cluster know exactly when they will transmit and receive messages [7].

As mentioned earlier in this section, FlexRay inherited most of its features from TTP [33]. Therefore its error detection mechanism is the same as in TTP, which makes it fault tolerant and safe for safety-critical real-time communication.

However, since the static segment in a TDMA round of FlexRay is the same as in TTP, allocation of static and equal length time-slots to all the nodes irrespective of their transmission payload can cause inefficiencies regarding channel utilisation, as nodes with shorter transmission payload cannot utilise the full length of their allocated time slots. FlexRay does not support any dynamic transmission schedule that can allocate different length time slots to the same node during different modes of operation depending upon the transmission payload of that node.

To improve the network utilisation in FlexRay, an algorithm is proposed in [76] to obtain the optimal length of static messages, however, the messages that are longer than optimal length are migrated to the dynamic segment of FlexRay [76]. Therefore, shifting the safety-critical messages to dynamic segment removes them from the safety-critical domain and compromises the reliability of the protocol. A heuristic algorithm is proposed in [77] to efficiently utilise the bandwidth of a FlexRay network. The basic idea is to utilise both channel in FlexRay independently [77]. Two modes have been introduced, called independent mode and fault-tolerant mode [77]. It is assumed that some frames in static segment do not need fault tolerance and hence, the independent mode can be used to send different messages on both channels in a given time slot [77]. This idea is in contradiction of the basic theme of a static and dynamic segment of FlexRay and therefore, does not provide an optimal solution if all frames scheduled in the static segment require fault-tolerance properties. The work of Lee et al. [78] to avoid transient failures in FlexRay by introducing retransmission of frames in static segment [78] further reduces the bandwidth utilisation and is prone to replicate communication errors. Similarly, other recent work such as [79] only addresses the issue of computation of the end-to-end delay for the messages those are scheduled with slot multiplexing in dynamic segment of FlexRay.

The deterministic communication in TTP and FlexRay therefore comes at a cost. Although communication is guaranteed to be collision free and existing TTA protocols have a known channel access latency, they have the disadvantage of poor channel utilisation. As each node has different functionality and consequently different transmission payload requirements, allocating equal-length slots to all nodes inevitably causes inefficiencies, as nodes with lower payload will not be able to fill their allotted time slots with meaningful data.

3.6 Time Triggered CAN (TTCAN)

Time-Triggered CAN (TTCAN) [80] was developed on top of the physical layer of the widely used, event-triggered CAN protocol [80]. The idea was to develop a flexible, hybrid protocol that can transmit time-triggered as well as event-triggered messages [80]. The protocol uses an exclusive window to transmit a safety-critical message that needs a guaranteed latency [80]. Unlike the original CAN protocol, safety-critical messages are transmitted at specific points in time (by using exclusive windows) and do not need to compete for bus access with messages transmitted using the CAN arbitration protocol [80]. The system matrix of TTCAN consists of a number of basic cycles and it allows TTCAN to choose multiple sending patterns, e.g., transmit a message once per basic cycle, once in a whole matrix cycle, etc [80].

A master node concept is used to synchronise clocks of all participating nodes [80], but this mechanism can add a significant delay in choosing a new master node in case of failure of the active master node. TTCAN does not provide important dependability services at the protocol level such as membership, independent bus guardians, reliable acknowledgment, or similar. Some, but not all of these services can be build at the application level, but at the expense of the efficiency of the protocol and timing bounds.

3.7 Time Triggered Ethernet (TTEthernet)

TTEthernet [81] was developed to enable time-critical real-time traffic over a standard Ethernet network. It supports three classes of traffic, Time Triggered (TT), Rate Constrained (RC), and Best Effort (BE) [81]. The schedule is computed offline for TT traffic, hence guarantees contention-free communication over the same network [81]. A transmitter node sends TT messages in pre-defined, static time slots in order to avoid collisions, however these slots are distributed over equal-size communication cycles, repeating indefinitely [81]. In TTEthernet, end systems (nodes) are connected through switches [81]. Flows (frames) can be transmitted from one end system to multiple end systems through these switches [81]. TT frames are periodically transmitted in pre-assigned time slots [81]. A transmission slot from one node, say Node A to Node C through switch 1, may be different in duration than the time slot from Node B to Node C through the same switch but remains the same (static and equal length) along the same path [81].

Each node in the TDMA round of TTEthernet may have a different slot length and the TDMA round cyclically repeats [81]. This means that the length of TDMA rounds across the cluster cycle are required to be the same. Therefore, TTEthernet does not cover more dynamic scenarios where a node requires different slot lengths in different TDMA rounds. Moreover, fault tolerance is achieved through redundancy management and does not cover all fault scenarios at the protocol level, missing important services such as membership, implicit acknowledgment, overhead-free, fault-tolerant clock synchronisation, clique avoidance and the like. Similarly, the bus guardian mechanism does not support variable slot lengths for the same node over different TDMA rounds.

3.8 Audio/Video Bridging and Time Sensitive Networking

Current audio and video encoders and decoders can generate frames that vary in size by order of magnitude [82]. The Audio/Video Bridging (AVB) [83] Task Group developed a set of protocols to support the deterministic communication of audio/video (AV) streams over a standard Ethernet network [84]. Despite the advantages of standard Ethernet such as high bandwidth and low cost, it does not provide temporal properties that are essential for real-time traffic. The AVB Task group introduced a set of standards such as 802.1As [85], 802.1Qat [86], 802.1Qav [87], and 802.1BA [88] to support low latency and jitter requirements for multimedia streams. Time synchronisation is supported by the 802.1AS standard, which is based on the IEEE1588 Precision Time Protocol (PTP) [89]. The Stream Reservation Protocol (SRP), also known as 802.1QAT is utilised to reserve the bandwidth for high-priority traffic classes, while 802.1QAV supports a queuing and forwarding policy for AV traffic [90]. AVB was introduced to provide low latency and jitter for AV traffic, by reserving bandwidth along the whole path from transmitter to receiver [84].

Despite its success and widespread use in the automotive industry, AVB fails to provide the real-time and fault-tolerance capabilities to support the rigid timing requirements of hard real-time applications [91].

Improvements have been made to IEEE AVB standards by the Time Sensitive Networking (TSN) Task group [92] in order to support real-time capabilities and performance improvements. TSN introduces different standards that are built on top of the AVB standards [92]. The Credit-based Shaping (CBS) algorithm used by the IEEE AVB 802.1Qav [87] standard does not support timing requirements of TT streams of AV traffic when non real-time traffic is in transmitted over the same channel [90]. CBS is used to overcome the issue of starvation for low priority traffic, but due to its non-preemptive nature, a low priority AV stream can block the transmission of time-critical AV streams [93]. TSN introduced the Time-Aware

Shaper (TAS) in IEEE 802.1Qbv [94] to resolve this issue. TAS adopts a preemptive approach where scheduled traffic can preempt low-priority traffic to fulfil its timing requirements [94]. TSN defines high-priority queues for TT traffic while the rest of the queues are same as used in AVB [94]. The traffic that does not require strict temporal properties is categorised as best-effort and assigned the least priority [95]. TSN is using IEEE 802.1AS-Rev [96] to synchronise the clocks to form a global time base for enabling the deterministic communication [96]. However, this mechanism comes at the cost of extra overhead (synchronisation frames) in addition to the normal traffic over the communication network [97]. TSN uses the concept of Gate Control List (GCL), which is implemented on the egress ports of each participating device in the network [97]. Each port can have multiple queues, where some of the queues are assigned to TT traffic and the rest of the queues are assigned to other traffic types such as AV or BE traffic [97]. GCLs are computed offline and at each egress port, frames will be transmitted from a queue whose gates are opened. When gates for TT queues are opened, the gates for other queues must be blocked [97]. It is to be noted that when a TT queue has multiple frames and its gate is opened based on GCL then a FIFO mechanism is used to transmit the frames from the same queue [98]. This may lead to unavoidable delays in transmitting safety-critical information over the network as fragmenting a flow into numerous frames and adding sequence numbers need extra time [97]. The complexity increases when there are multiple hops between end systems [97]. A GCL scheduled for a priority queue defines the exact interval when that queue has exclusive access to the transmission channel [97]. Interleaving frames from different TT flows to the same priority queue can significantly increase end-to-end transmission delays [99]. To avoid arbitrary transmission of TT frames and handling of babbling idiot faults, TSN introduces time-based ingress policing in IEEE 802.1Qci [100]. A time-aware Access Control List (ACL) is used to keep track of the arrival time of incoming TT frames. The ACL is computed offline and must be aligned with GCLs [100]. A TT frame can be transmitted successfully only if the ACL grants permission to pass at the ingress port and at the same time the GCL has an active transmission time slot at the egress port [100]. The GCL period is computed offline for a TT flow and repeats in cycles, which means the GCL has a static and equal-length time period for a TT flow as used in [99]. TSN introduces reliability and fault tolerance by using IEEE 802.1CB [101]. A transmitter or any intermediate device such as a switch will generate sequence numbers for all frames and multiple copies are generated for each frame before transmitting them over the network [101]. Therefore, to identify and control the individual streams or flows, additional mechanisms such as Per-Stream Filtering and Policing [102] as well as Frame Replication and Elimination for Reliability [103] are required. To avoid a single point of failure, redundant routes are configured by using IEEE 802.1Qca [104] and copies of a frame are transmitted over these routes[104]. The issue of network overloading is resolved by eliminating the duplicated copies of the frame, either by an intermediate device, such as a switch, or at the receiver end system [97]. The TSN fault-tolerance mechanisms introduce additional complexity and latency, which is not suitable for low-latency SCRTs [97]. Most of the TSN approaches [105] use uni-casting to transmit TT flows, therefore advantages of membership service such as atomic multicast, tracking the status of active and inactive nodes in the cluster, clique avoidance, and implicit acknowledgment without any extra overhead cannot be implemented at the protocol level in the TSN standards [105]. The most complex issue with TSN is the computation of its TT communication schedules for a large number of network components [105]. Interdependency of routing and communication schedules become computationally intensive due to combinatorial explosion [97]. Recently, a number of TT schedulers [106][107][108] have been proposed to solve the aforementioned issues sequentially and some approaches [109][110][111] use ILP-based solutions but are very time consuming and not scalable for large real-time systems [97]. Most of the TT schedulers [106][107][109][110][111][112] for TSN assume that the underlying network infrastructure is fault-free. In reality such assumptions don't hold for SCRTs. Consequently, the TT scheduling problem under faults environment can lead to a computationally intractable scheduling process [97]. In addition, although TTEthernet and TSN are aimed at using existing, low-cost Ethernet infrastructure for timing-critical traffic, their implementation comes at a high cost of maintenance and design complexity [113].

3.9 Wireless communication for safety-critical clusters

Wireless communication techniques are being actively developed to be used in safety-critical real-time systems such as Intelligent Transport Systems (ITS) [114–116]. Numerous car manufacturing and telecommunication companies as well as research and development institutions worldwide, are working to develop a variety of vehicular communication networks [117] [118][119]. With the advances in the technology, vehicles are becoming increasingly smart. Vehicle connected with each other and with Road Side Units (RSUs) allow updates about weather, traffic density on routes, and communication of safety-related information to other vehicles and surrounding infrastructure [120]. Such systems are referred in the literature as vehicle-to-vehicle (V2V), vehicle-to-infrastructure (V2I), vehicle-to-everything (V2X), and vehicular ad hoc networks (VANETs) [117] [118][119]. The most commonly used communication protocol under the V2X architecture is 802.11p, defined in [121] and revised in [122]. Channel access in 802.11p is based on a Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) scheme, where all transmitter nodes participate in channel sensing before starting their transmission [122]. On sensing the channel as busy, a node will delay its transmission by choosing a random backoff value [122].

The unsynchronised channel access mechanisms in these wireless architectures can result in significant transmission delays. This is not suitable for safety-critical real-time systems by any means. Another MAC method proposed for VANETs is Self-organising Time Division Multiple Access (STDMA) [123], where time is divided into time slots constituting a frame which means a node is transmitting its frame in multiple time slots [123]. For example, a case study used in [124] is using 904 and 2283 time slots per frame in its two different data traffic models. It is important to note here, that any wireless protocol is subject to potential interference with associated packet loss or unbounded delays and thus is considered a best-effort protocol [123] [122], not meeting the fault-tolerance requirements of safety-critical real-time systems.

Therefore, using wireless communication in safety-critical real-time systems such as in-vehicle networks to connect different nodes of a safety-critical cluster such as an anti-lock braking (ABS) system is not an appropriate choice as these protocols are not capable to cover all the fault scenarios considered in this thesis. However, it is possible that these protocols can play an important role in augmenting on-board systems. For example, a gateway can be designed to link safety-critical clusters of a wired network with a wireless network such as V2X architecture to exchange the information on road infrastructure, congestion, and the like.

Chapter 4

Dynamic Time-Triggered Communication

In distributed safety-critical real-time systems, different nodes are connected with each other through a shared communication channel such as bus. They coordinate their actions through message passing, therefore, timely and reliable message delivery is critical [2]. Communication errors and unpredictable delays in transmission may lead to unpredictable behaviour of distributed real-time systems [2]. Let us examine the brake-by-wire system in a car we discussed earlier. When the driver hits the brake pedal, then, based on physical parameters such as the speed of the car and its wheels, a brake force is calculated and transmitted to each wheel for stopping the car [2]. A slight delay or error in communication may lead to a longer stopping distance or even complete brake failure, potentially causing harm [2]. To avoid certain delays to access the communication channel, Time-Triggered Architecture (TTA) uses a Time Division Multiple Access (TDMA) scheme for channel access [6]. Communication time slots used by the nodes that make up the distributed system are deterministic and scheduled offline [6]. To this end, these communication time slots are static and equal-length for all nodes in a TDMA round irrespective of their transmission payload requirements. The same communication slots repeat over different TDMA rounds in a cluster cycle. However, this static scheme comes at the expense of flexibility, which results in poor channel utilisation, as we will see. This chapter develops an alternative approach that utilises configurational flexibility in allocating the slot length of each node on the basis of its transmission payload requirements, while retaining the properties required for safety-critical operation.

4.1 Motivation

Flexibility and dependability are the two parameters that are often considered as contrary to each other, and choosing between them to solve an engineering problem is a hard task [22][23]. There is a strong argument in the literature [6] [33] [68] that to achieve and verify dependability, static prior knowledge about the sequence and timing of state changes is essential [2]. This idea has been exploited by the Time-Triggered Architecture (TTA) [6][5] to produce more dependable real-time systems [125]. In TTA-based communication, permission to use a communication channel is determined by a Time Division Multiple Access (TDMA) scheme with pre-defined time slots [6]. The most prominent example of such communication in real-time systems is the Time-Triggered Protocol (TTP) [2][8]. To ensure collision-free channel access, all nodes are assigned time slots to transmit their information [6]. All nodes have synchronised clocks to form a global time base, and thus, they know the exact point of time each node will transmit its message [6][33][68]. All static time slots are of equal duration and are distributed between participating nodes in equal-length TDMA rounds. These TDMA rounds repeat indefinitely, making channel access periodic.

This predictability ensures that channel access will be inherently free of collision and all the protocols that follow this approach will have known channel access latency [6]. However, this comes at a cost. In a system, it is likely that different nodes have very different transmission requirements for their transmission payload which can vary in each TDMA round of a cluster cycle. Therefore, they need different length time slots, and assigning equal length time slots will result in poor channel utilisation (CU).

There are a number of approaches [70, 71, 99, 126–131] that make an effort to overcome this issue. None of these approaches completely solve this problem while also retaining the requirements that are imperative for Safety-Critical Real-Time (SCRT) systems, such as fault tolerance and guaranteed timeliness. In fact, this problem has long been recognised and attempts exist to split up communication and, at least, provide some flexibility and better channel utilisation for non-essential messages, while retaining the strict real-time guarantees for safety-critical communication [7]. Time-triggered scheduling and communication models [132–134] are implemented in different real-time applications but none of them use a flexible slot allocation that retains the fault-tolerance features required at protocol level.

4.2 Payload based slot lengths inside a TDMA round

This section discusses **INCUS** (Individually Node Slot CUStomizable protocol), my approach to overcome the above-mentioned limitations towards efficient channel utilisation in time-triggered communication. INCUS is a protocol developed in the course of this research that allows the slot length of nodes to be configured in accordance with their payload requirements when a TDMA round is designed. The feasibility of this proposed approach is evaluated while retaining the level of reliability required for safety-critical real-time systems. An analysis in this chapter will show an almost twofold improvement in efficiency in a typical automotive, brake by wire scenario.

All the existing TTA communication protocols for safety-critical, distributed real-time systems use a TDMA approach for channel access where static and equal length node slots are used in each TDMA round for transmitting the safety critical information as shown in Figure 4.1. Here, the term *traditional approach* is used for such an approach.

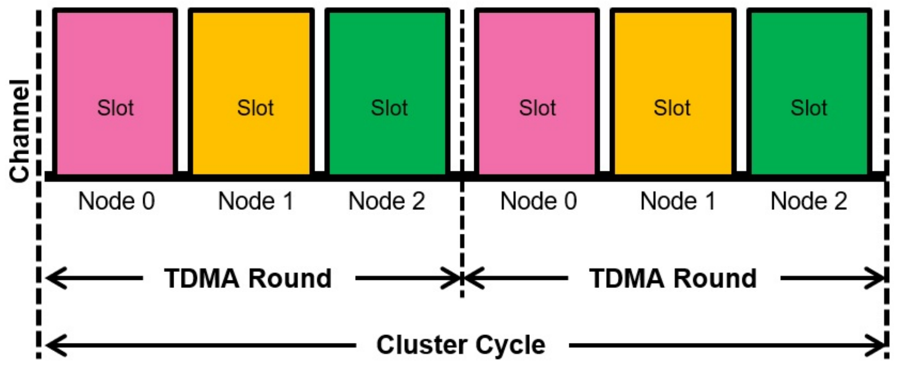

FIGURE 4.1: TDMA based approach for channel access

I would argued that, while care must be taken to facilitate fault tolerance and ensure replica determinism, assigning same-length node slots will cause low channel utilisation and will affect the efficiency of the communication protocol. This is due to each node in the system typically having different functionality and, thus, associated transmission requirements. The slot length of each node should be configured according to its transmission payload requirements in a TDMA round and may repeat in subsequent TDMA rounds as shown in Figure 4.2. The Brake-by-Wire (BBW) case study [29] is used to illustrate this problem further.

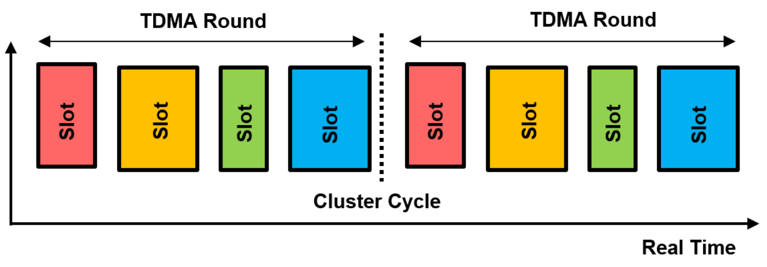

FIGURE 4.2: INCUS based approach for channel access

4.2.1 Brake-by-Wire Case Study

X-by-wire systems such as steer-by-wire, brake-by-wire, and the like, are nowadays starting to become more prevalent in the automotive industry [29]. These systems are designed to replace mechanical components with more sophisticated electrical components including sensors and electrical actuators [29]. The goal is to make these systems more reliable and fault tolerant than traditional mechanical systems (and it is expected for them to also become cheaper, once used in a higher number of mass-produced vehicles) [29].

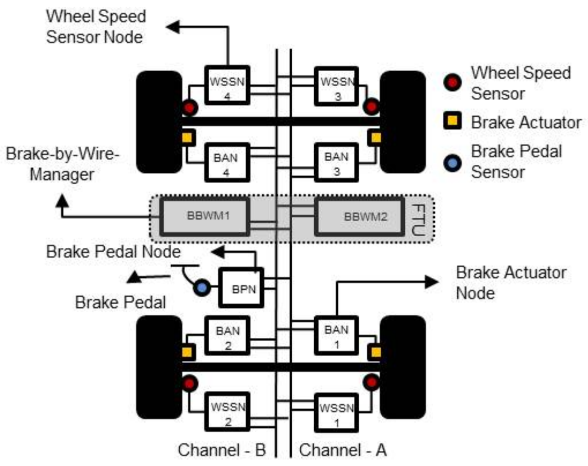

Figure 4.3: Brake-by-wire Architecture

The Anti-lock-Braking System (ABS) widely used in modern vehicles is an example application that can sit on top of a brake-by-wire system [29]. ABS provides more safety to drivers by avoiding wheel lock-up and uncontrolled skidding [29]. It also decreases the stopping distance on dry and slippery roads. The layout of a typical brake-by-wire system is shown in Figure 4.3. Each wheel has a wheel speed sensor and a brake actuator. The function of the Brake-by-Wire-Manager (BBWM) is to use the brake pedal sensor value from the pedal node (BPN) and the speed of each wheel from the wheel speed sensor nodes (WSSNs) to calculate the brake force for each brake actuator node (BAN). Each BAN will take this brake force value and apply it to their corresponding wheel through their brake actuators. The BBWM periodically monitors the wheel speeds reported by the WSSNs and checks the difference between them. If a wheel is about to lock, the BBWM will send a lower brake force to that wheel's BAN. Wheels that spin faster than others will have a stronger brake force applied. To provide fault tolerance at the hardware level, two redundant BBWM are used. However, the WSSNs and BANs are not replicated, as it is possible to brake the car with the remaining wheels, even if one wheel unit fails. The whole communication among BBWM, BPN, WSSN and BAN is based on replicated communication channels with a bandwidth of 1 Mbps. The BAN for

Table 4.1: Frame Sizes for the Nodes of the Brake-by-Wire System

Node	**Data**	**Frame Size**	**Slot Length**
Node	Data	Frame Size	Slot Length
BBWM	12 bits	$12 \cdot 4 + 28 = 76$ bits	76 μs
BPN	10 bits	$10 + 28 = 38$ bits	38 μs
WSSN	10 bits	$10 + 28 = 38$ bits	38 μs
BAN	nil	$0 + 28 = 28$ bits	28 μs

each wheel is not transmitting any application data, but control information to show its membership on the network is still transmitted.

Table 4.1 shows the payload data and ideal transmission lengths for all the nodes in the Brake-by-Wire system. The transmission lengths also take into account 28 bits of control information with the same functionality as in TTP [33] (a 1-bit frame identifier, 3 bits of mode information, and a 24-bit CRC value). Each type of node in the BBW system requires a different node slot length. If a traditional TTA slot allocation approach is used, the slot length for each node will need to be the maximum 76 microseconds required by the BBWM. All other nodes transmit only half the data or less and would have to pad their node slot to the static, maximum length. No other node can transmit during that padding time, so this is considered as the node slot overhead time as illustrated in Figure 4.4.

In INCUS, the slot length for each node is configured in accordance with the transmission requirements of a node. Rather than allocating same-length slots to all nodes, each slot length is configured on the basis of its transmission data requirements. Consequently, node slots vary in their length within a TDMA round as shown in Figure 4.5. This concept eliminates the overhead time of a node during transmission and improves the overall efficiency of channel utilisation (as shown in Section 4.2.5).

The principle of operation of INCUS is otherwise based on TTP [33]. The communication controller subsystem of a node that is actually responsible for transmission and reception of data over the communication channel operates under similar principles. The *Message Descriptor List* (MEDL, a data structure within controller memory) holds the time schedule for the data transmission and data reception

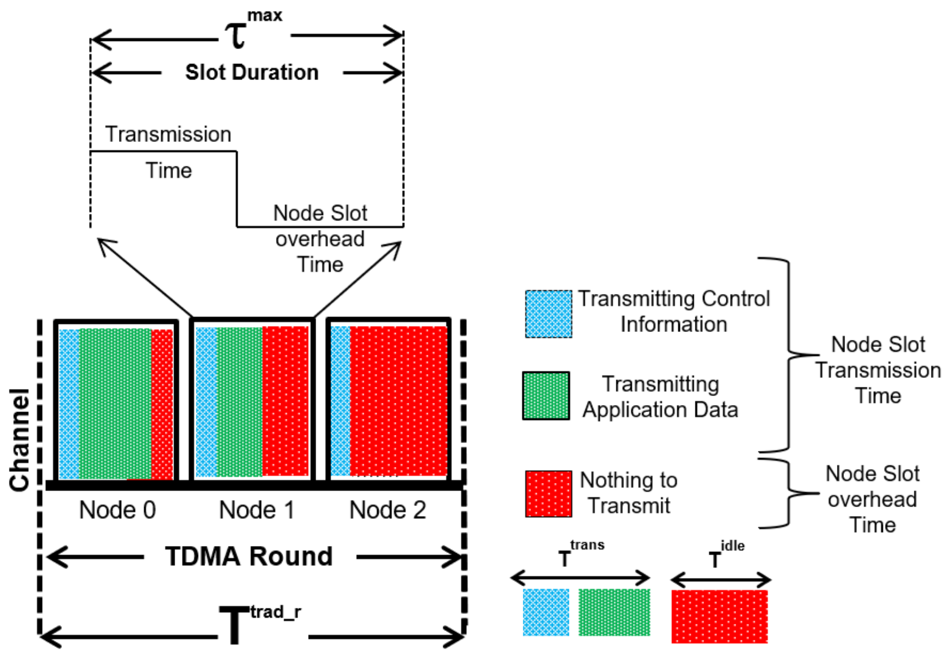

FIGURE 4.4: Traditional slot allocation approach

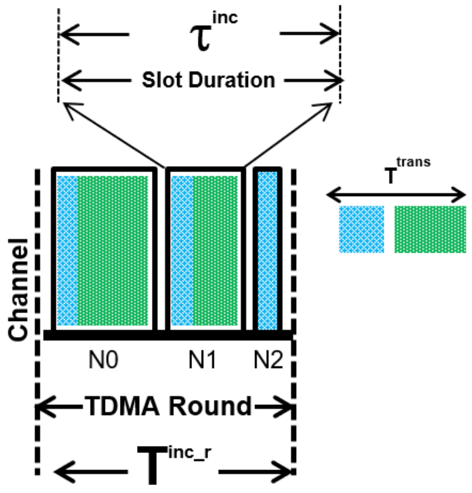

FIGURE 4.5: INCUS slot allocation approach

phase for all the nodes. MEDL used in INCUS is different than MEDL [8] as it holds the transmission schedule of each slot of a node in all TDMA rounds (slot length vary in each TDMA round). Each node has a replicated copy of the MEDL, hence it knows the exact time when it and other nodes have the right to access the communication channel. Depending on the nature of transmission, three classes of frames are used. Normal frames (N-frames) carry application data, initialisation frames (I-frames) carry synchronisation information for reintegration of recovering nodes, and coldstart frames (CS-frames) carry synchronisation information for the integration of nodes during system start-up [8].

4.2.2 Node Startup and Resynchronisation

Node startup and resynchronisation is based on TTP [8], but INCUS requires different values of timeout parameters. All nodes have a unique listen timeout (time for a node to wait for I-frame) and coldstart timeout (time for a node to wait for a response from other nodes on the network after transmitting a CS-frame), because of their unique startup delay defined in Equation 4.1.

$$T_0^{start} < T_1^{start} < .... < T_i^{start} < ... < T_{n-1}^{start} \tag{4.1}$$

In INCUS, a startup delay for a $node_i$ is the duration of all *incus-slots* from the beginning of the *incus-round* up to the beginning of the *incus-slot* of $node_i$. The term T^{inc_r} (*incus-round*) denotes a TDMA round in INCUS and τ^{inc} (*incus-slot*) represents a specific node slot. It is important to note that this differs from TTP in that $\tau_i^{inc} \neq \tau_j^{inc}$ is now possible! In TTP, the listen timeout parameter for a recovering node i during the integration process is $T_i^{start} + 2T^{trad\_r}$ with T^{trad_r} being the time for a traditional TDMA round [8]. This may result in the frequent transmission cold-start frames in case node i did not receive any I-frames for up to two traditional-rounds. In INCUS, I bound the listen timeout by the number of incus-rounds in a cluster cycle.

$$T_i^{listen} = T^{c\_cycle} + T_i^{start} \tag{4.2}$$

where the cluster cycle time for r incus-rounds is

$$T^{c\_cycle} = \sum_{i=1}^{r} T_i^{inc\_r} \tag{4.3}$$

such that the incus cold-start time is

$$T_i^{inc\_cold} = T^{inc\_r} + T_i^{start} \tag{4.4}$$

I-frame Transmission. TTP has to transmit I-frames on both channels after two traditional rounds [8]. INCUS can accommodate a more flexible approach in transmitting I-frames. For example, if there are some nodes, such as actuator nodes in the network, that are not transmitting any data at all (other than control information to show their presence on the network) then I-frames can also be transmitted by these nodes. This may be desirable for faster reintegration of recovering nodes, as they will receive I-frame in each incus round. Conversely, any two nodes can be statically configured to transmit I-frame only once, in each cluster cycle as defined in Equation (4.3), transmit I-frames on both channels, reducing the overhead from I-frame transmissions.

4.2.3 Membership Service

To facilitate fault tolerance and keep track of active and inactive nodes within a round, a TTA membership service [8, 68] is implemented, by recording the membership status of each node in a membership vector whose bit size is equal to the number of nodes in the cluster. It is important to note that, following the TTA [6], this comes without any additional acknowledgement overhead or timeout requirements, as each node transmits once per incus-round. For every slot, the transmission of a node (or lack thereof) is registered in the membership vector. As in TTP, the membership vector does not have to be transmitted explicitly [8], but is used in the frame CRC calculation, causing dissenting minority nodes (those with a differing view from the majority) to no longer be heard (and no longer being able to receive frames from the majority of nodes within the cluster) [8]. The membership vector of such minority nodes will quickly drop to half the nodes in the system or below, causing them to restart and re-integrate into the cluster [33]. A detailed mechanism of membership service is discussed in Section 4.3.2.1.

4.2.4 Clock Synchronisation

Clock synchronisation [68] is a core requirement of time-triggered communication protocols [6, 33]. All the nodes have physical clocks (internal or local clocks) and a predetermined schedule defined in their own copy of the MEDL [8]. Each node operating in the system times its slots (transmit or receive actions) in accordance with a predetermined schedule that is triggered by the progression of time [6, 68]. For these node slots to be synchronised, the nodes' clocks have to be synchronised. Clock synchronisation in INCUS is based on TTP [8] and follows the same principles. Hence, clock synchronisation in TTP will be described first before discussing the differences in INCUS. Any clock skew between sender and receiver is identified at the receiver side by using the difference between the expected arrival time defined in the MEDL and the actual arrival time of a frame [8]. The slots that are to be used for clock synchronisation are denoted in the MEDL (e.g., transmissions from nodes with cheaper, less accurate clocks may be disregarded) [8]. These slots are called resynchronisation points R_p. The Fault Tolerant Average (FTA) [59] is then used to correct the local clock based on the measured deviations. There are two phases to perform clock synchronisation [59]. In the first phase, the difference is stored in a sorted array with a size of four where the highest and lowest values are discarded and then the average of the remaining two values is calculated [59]. This average value is used as the correction term to adjust the local clock's time [59]. If resynchronisation point is set after n number of slots then duration of R_p can be calculated in TTP [33] as follows: let t be the real time such that:

$$R_p(t) = n * \tau^{max} \tag{4.5}$$

where τ^{max} is the node slot length in TTP.

As in TTP [33], all nodes have the same length time slot to access the communication channel, the resynchronisation interval can be calculated by using Equation (4.5). For INCUS, the nodes can have different length slots, so it need to adjust the formula as follows:

$$R_p(t) = \sum_{i=0}^{n-1} S_i(t) \tag{4.6}$$

where $(n \geq 4)$ and $S(t)$ is the slot length of each node.

With the membership and distributed clock synchronisation mechanism, INCUS provides the basis for fault tolerance through node replication and ensures replica determinism. Use of duplicated communication channels allows us to tolerate a single channel failure. As the communication schedule of all the nodes is predefined, the additional mechanisms in the TTA can be used, such as bus guardians [8], to prevent a "babbling idiot" node from transmitting outside its allotted slot [6]. Hence, in terms of fault tolerance and other safety critical features, INCUS is on par with TTP.

4.2.5 Efficiency Analysis for Channel Utilisation

In order to analyse the channel utilisation in the existing traditional approach and compare it with INCUS, there is a need to define the timing terms to perform the analysis.

- T^{trad_r} represents (the length of) one TDMA round using the traditional slot allocation approach.
- T^{inc_r} represents one TDMA round in the proposed slot allocation approach for INCUS.
- τ^{max} represents the node slot length in the traditional slot allocation approach (same for all the nodes).
- τ_i^{inc} represents the slot length for each node in INCUS (possibly different for each node i).
- T_i^{trans} is the transmission time for control and payload data for node i during its allocated node slot.

- T_i^{idle} is the remaining allocated slot time for node i not utilised for data or control information.
- T^{ifg} is the Inter Frame Gap (IFG) overhead time, i.e., the time when no transmission occurs between frames.
- T_i^{ovhd} is the total overhead for node i for its allocated slot.

4.2.5.1 Traditional slot allocation approach

Due to equal length slot allocation to all the nodes, it can say that slot length (τ^{max}) of each node consists of node slot transmission time (T_i^{trans}) and a node slot overhead time (T_i^{ovhd}) as shown in Figure 4.4. Node slot overhead time of a $node_i$ can be calculated as:

$$T_i^{idle} = \tau^{max} - T_i^{trans} \tag{4.7}$$

$$T_i^{ovhd} = T_i^{idle} + T^{ifg} \tag{4.8}$$

If T_i^{ovhd} is the overhead time in each node slot then total overhead time ($\hat{T}^{ovhd}$) for $\boldsymbol{n}$ number of nodes in a TDMA round is:

$$\hat{T}^{ovhd} = \sum_{i=0}^{n-1} T_i^{ovhd} \tag{4.9}$$

If there are $\boldsymbol{n}$ number of nodes then the length of a TDMA round (T^{trad_r}) in traditional slot allocation approach is defined as:

$$T^{trad\_r} = n \cdot (\tau^{max} + T^{ifg}) \tag{4.10}$$

By using Equation (4.9) and Equation (4.10) channel utilisation (CU) in a TDMA round for traditional slot allocation approach can be calculated as:

$$CU = \frac{T^{trad\_r} - \hat{T}^{ovhd}}{T^{trad\_r}} \tag{4.11}$$

4.2.5.2 INCUS approach

In the proposed protocol, the slot of each node is customised on the basis of its transmission load to avoid the node slot overhead time within the allocated node slot as shown in Figure 4.5. Therefore, for slot length τ^{inc} of each node, T_i^{idle} is zero and the slot length of a $node_i$ is:

$$\tau_i^{inc} = T_i^{trans} \tag{4.12}$$

If there are a number of n nodes then length of a TDMA round (T^{inc_r}) in my slot allocation approach can be defined as:

$$T^{inc\_r} = \sum_{i=0}^{n-1} (\tau_i^{inc} + T^{ifg}) \tag{4.13}$$

Channel utilisation in a TDMA round for the this approach can be calculated by using Equations (4.9) and (4.13):

$$CU = \frac{T^{inc\_r} - \hat{T}^{ovhd}}{T^{inc\_r}} \tag{4.14}$$

As there is no node slot overhead time in my approach, every node will fully utilise its allocated node slot time for transmitting data and control information over the communication channels. Hence, INCUS can transmit the same amount of data in less time than the traditional approach. In the exceptional case where the payload requirements for all the nodes are the same, traditional slot allocation overhead is equal to INCUS. In all other cases, INCUS avoids the additional overhead caused by unequal payloads.

4.2.5.3 System-Level Analysis

Now let's apply the analysis regarding channel utilisation to the Brake-by-wire case study. There are a total of eleven nodes connected through a bus[1] as shown

[1]For simplicity, we ignore the replicated bus here – however, it is important to note that the timing requirements for the redundant bus are exactly the same.

Table 4.2: Channel Utilisation for the Brake-by-wire System using Traditional Slot Allocation

Node	τ^{max}	T_i^{ovhd}
$\text{BBWM}_{1,2}$	76μs each	$(0+4)\cdot 2 = 8\mu$s
BPN	76μs	$(38+4)\cdot 1 = 42\mu$s
$\text{WSSN}_{1\ldots4}$	76μs each	$(38+4)\cdot 4 = 168\mu$s
$\text{BAN}_{1\ldots4}$	76μs each	$(48+4)\cdot 4 = 208\mu$s
Total	$880\mu s$	$426\mu s$

Table 4.3: Channel Utilisation for the Brake-by-wire System using INCUS Slot Allocation

Node	τ^{inc}	T_i^{ovhd}
BBWM1-2	76μs each	$(0+4)\cdot 2 = 8\mu s$
BPN	38μs	$(0+4)\cdot 1 = 4\mu s$
WSSN1-4	38μs each	$(0+4)\cdot 4 = 16\mu s$
BAN1-4	28μs each	$(0+4)\cdot 4 = 16\mu s$
Total	$498\mu s$	$44\mu s$

in Figure 4.3.

According to the traditional slot allocation approach the slot length τ^{max} of each node should be 76μs. Only two nodes, BBWM_1 and BBWM_2 will fully utilise that slot length for transmitting information over the communication channel; the rest of the nodes will have a node slot overhead time in their allocated slot length as shown in Table 4.2. INCUS is using 4 μs for T^{ifg}, therefore a slot length, plus T^{ifg}, is 80 μs for each node. According to Equation (4.10) T^{trad_r} is 880μs where $\hat{T}^{ovhd}$ by using Equation (4.9) is 426 μs. By substituting these values in Equation (4.11), channel utilisation for a TDMA round according to traditional slot allocation approach is **51.59%**. In the traditional slot allocation approach, the 454 bits of payload data are transmitted in a TDMA round of 880 μs.

By comparison, in INCUS, the slot length τ^{inc} of each node depends on its transmission requirements as shown in Table 4.3. Therefore, T_i^{ovhd} is zero for all nodes. As T^{ifg} is 4 μs and T_i^{idle} is 0 μs, according to Equation (4.9) $\hat{T}^{ovhd}$ is only a total of $44\mu s$ in INCUS.

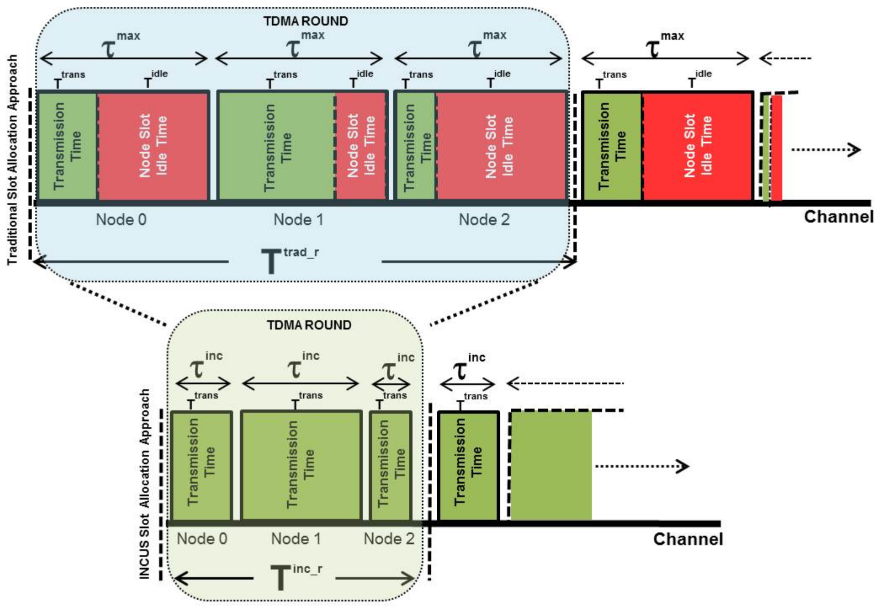

FIGURE 4.6: INCUS vs Traditional approach

This constitutes only 10.3% of the overhead of the fixed maximum-length slot allocation approach shown before. Consequently, following Equation (4.13), T^{inc_r} now totals 498μs per TDMA round (compared to 880μs in a traditional protocol such as TTP). By substituting these values in Equation (4.14), the channel utilisation for each TDMA round is **91.16**% when using INCUS slot allocation. Figure 4.6 illustrates the comparison between the traditional and INCUS slot allocation approaches.

4.3 Flexible communication schedules in different TDMA rounds

Dependable SCRT systems are becoming increasingly important and complex. Examples of such systems are autonomous or self-driving cars, which are poised to revolutionise the transport industry. A critical part of these SCRT systems is the network communication protocol that is used by components in an SCRT system to exchange data. Communication protocols for SCRT systems are required to exhibit predictable, worst-case execution times and thus have to be designed in a more static [2] and less flexible way. To ensure this predictability, current state-of-art communication protocols for SCRT systems are based on the Time-Triggered Architecture (TTA) [6], where static and equal length time-slots are used for all nodes to access the communication channel, irrespective of the size of their transmission payload. This determinism forms the basis of predictable timing, behaviour and fault tolerance [6]. However, as we saw, this determinism comes at the cost of poor channel and bandwidth utilisation, which hinders the development of SCRT systems.

In the previous sections, INCUS has shown that assigning time slots on the basis of transmission payload of each node in a Time Division Multiple Access (TDMA) round significantly improves the communication efficiency. This section will present an enhanced version of INCUS, INCUS+, where optimisation of communication schedules are proposed for different TDMA rounds of a cluster cycle.

INCUS+ allocates the slot length of a node based on its varying transmission requirements in each TDMA round of a cluster cycle. We will see that this flexibility can be achieved while retaining the level of dependability required for SCRT systems and still ensuring fail-silence. The INCUS+ design exhibits a significant improvement in bandwidth and channel utilisation, as will be demonstrated in an autonomous vehicle case study presented in this section.

4.3.1 Why dynamic communication over different TDMA rounds?

Let us investigate different use-cases that exhibit the need for different transmission slot length during different TDMA rounds of a cluster cycle. Nowadays Advanced Driver Assistance Systems (ADAS) are getting very common in modern vehicles [135]. While this was originally considered under the Telematics domain, it has, over time, been accreting more and more new features, to the effect that the ADAS is considered as a separate domain [135]. These systems are also communicating with in-vehicle safety networks to provide autonomous operations for the self-driving vehicle [135]. ADAS relies on videos collected from multiple cameras mounted on the vehicle [135]. Compressed videos from the camera are sent to ADAS through the in-vehicle safety networks [135]. Compression techniques such as H.264 produce frames of different data length and hence, a camera node may need a different transmission slot length in each TDMA round [135]. A number of similar scenarios are discussed in the literature such as [136] where the stream sender is using H.264 based compression technique for low latency video streaming for autonomous cars. Similarly, some approaches [137] use TDMA-based techniques to transmit compressed video frames but have to use multiple, equal-length time slots to transmit different-length frames that, consequently, results in poor channel utilisation.

Another use-case is Unmanned Aerial Vehicles (UAVs). These are increasingly used for numerous applications, including surveillance, exploring and tracking targets [138]. For example, UAVs are in high demand for the inspection of large scale structures as well as search and rescue operations in a disaster area [138]. One or more UAVs are used to transmit live video from an area of interest to a ground station where an operator can adjust the position of UAVs after analysing the streaming video [138]. In remote areas, multi-hop wireless networks are created where a number of UAVs are used as relays to extend the range [138]. Each relay (a UAV in this case) forwards a received packet to the next hop closer to the sink node [138]. This transmission is carried out by using a native wireless CS-MA/CA [14] arbitration scheme. Transient asymmetries between the relay links leads to unbounded packet buffering which further creates longer queuing delays and buffer overflow results in pack losses [138]. To overcome this issue, Pinto *et al.* [138] proposed a TDMA-based approach on top of standard WiFi, where an adaptive slot length for each relay node in every round mitigates the issue of unbounded queuing delays and reduces packet losses [138]. However, this approach was designed for soft real-time systems and does not provide any fault-tolerance guarantees.

The Internet-of-Things (IoT), is a new paradigm that is used to connect surrounding physical objects containing sensor or actuator nodes in order to operate them remotely through the Internet [139]. This concept is used in a number of applications from different domains such as healthcare, smart homes, etc [139]. Healthcare applications deployed in smart homes may consist of sensor devices to monitor the vital signs of a patient, alerting family members or physicians in an emergency situation [139]. Efficient communication is one of the biggest challenges in IoT as data networking is used to collect information (e.g. vital signs) from sensor devices [139]. Researchers have attempted to improve communication efficiency for such systems. Saxena [139] proposed a context-aware, adaptive, forwarding (Cdf) strategy to transmit critical, health-related data, even in poor network conditions. For example, if patient's vital signs are consistently good, the acquisition interval increases, which reduces the transmission rate of packets, otherwise, more frequent data transmission is required [139]. However, their approach to facilitate efficient data transmission is opportunistic [139], utilising a rules-based best-effort approach.

The objective of the proposed work is to overcome the issue of poor channel utilisation with a deterministic level sufficient for fault tolerance and predictability in SCRT systems. The principle of operation of the proposed protocol is discussed in Section 4.3.2. Section 4.3.3 discusses the proposed work in light of the need for *configurational flexibility* for slot allocation in TTA-based communication protocols, exemplified by a representative case study. In Section 4.3.4 a computational model is presented, comparing different slot allocation approaches. The proposed work shows how flexibility in slot allocation can significantly improve the channel and bandwidth utilisation.

4.3.2 The INCUS+ Protocol

In the literature (see Chapter 3), different communication protocols for safety-critical distributed real-time systems have been discussed. All the nodes in TTA-based communication protocols use static and equal-duration time slots for transmitting Safety-Critical Information (SCI) [7]. The term *FlexRay slot allocation* is used for the slot allocation mechanisms in such TTA based protocol i.e. FlexRay.

In this section, a slot allocation approach is presented that allows variable slot length of each node over the TDMA rounds of a cluster cycle as shown in Figure 4.7.

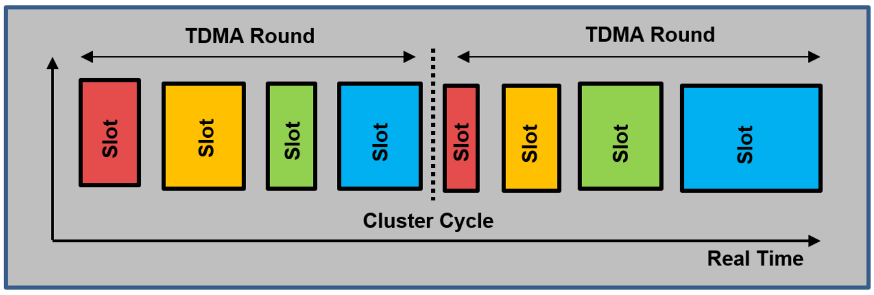

FIGURE 4.7: Slot length configurations during different TDMA rounds

We shall refer to this flexible approach as the *INCUS+ slot allocation* approach. Importantly, the slot length of each node is configured according to its payload requirements in each TDMA round, which means a node may have a different slot length in a different TDMA rounds. Consequently, the length of each TDMA round may vary in a cluster cycle. The need of such flexibility is justified by using an example of an autonomous vehicle case study (see Section 4.3.3).

The principle of operation of communication in our proposed approach follows [140] and [33]. The communication controller of each node is a subsystem that transmits and receives channel data and has a copy of the Message Descriptor List (MEDL) as shown in Figure 4.8. The MEDL holds information about different parameters including data transmission and reception time for each slot in each TDMA round of a cluster cycle for its respective node. The MEDL is statically configured and therefore each node knows the exact time to access the communication channel.

The three classes of frames used to control the communication operations over the communication channel are N-Frame, I-frames, and CS-frames as discussed in the previous section. Please note that in this and in the coming sections, the I-frame term is used for intra-coded pictures from the camera (used in case study, see Section 4.3.3.1), therefore, the full term *initialisation frame* is used instead of the usual TTA I-frame moniker that is colliding here with the video encoder frame type of the same name.

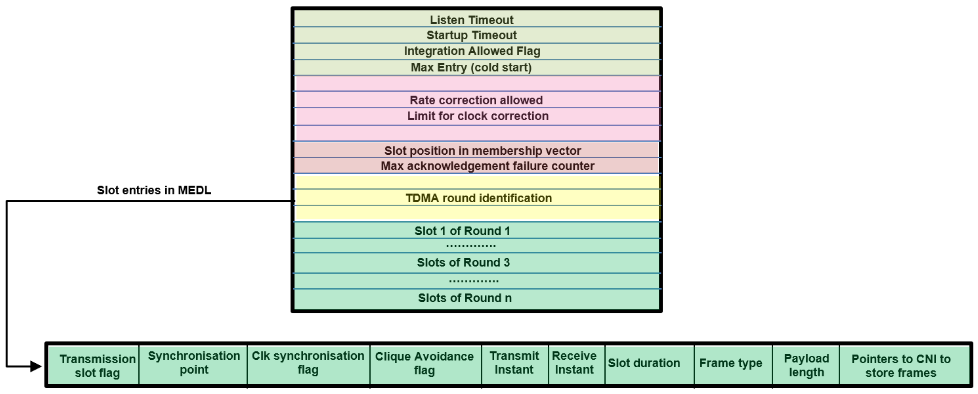

FIGURE 4.8: Layout of Message Descriptor List (MEDL) extended from [8]

The mechanism to start the protocol cluster or to reintegrate a lost node back to the cluster is the same as in TTP [8] but requires different timeout parameters. For $node_i$, the startup delay is the duration of transmission slots in the TDMA round up until the start of the $node_i$ transmission slot. Therefore, the start-up process for this proposed approach is much more efficient as it eliminates the node slot idle time. All nodes have unique parameters for the listen and coldstart timeout as they each have a unique start-up delay.

The configuration for the transmission of the CS-frame is same as used in INCUS which is bound with the number of TDMA rounds in a cluster cycle, unlike TTP where it is transmitted after each two TDMA rounds [8] that may result in frequent transmission of the frame if a node did not receive an initialisation-frame for up to two TDMA rounds. Initialisation-frames are used to reintegrate recovering nodes and typically, TTP is using a configuration of two TDMA rounds to transmit initialisation-frames over both replicated channels [8]. However, the process of reintegration can be improved if an actuator node (not transmitting any application data but control information) is configured to transmit initialisation-frames (See Section 4.3.3.3).

INCUS+ adds the flexibility necessary for better channel utilisation to time-triggered communication, while retaining the deterministic nature of the protocol regarding channel access. The communication schedule of each and every node is stored in the Message Descriptor List (MEDL), and each node has the copy of the MEDL. Therefore each node knows when to broadcast a frame over the communication channel and when to receive a frame from the communication channel. Please note that the term *flexibility* in this chapter is referring to configurational flexibility which will not impact the deterministic nature of the protocol. INCUS+ utilise the same fault tolerance features of earlier time-triggered protocols [33][7], such as distributed clock synchronisation, membership and acknowledgment service, bus guardians, and replica determinism to handle faults at the protocol level. The communication channel is prevented from being a single point of failure through the use of replicated communication channels.

4.3.2.1 Membership service and implicit acknowledgment

The membership service [8, 68] records the status of all nodes to facilitate fault tolerance. The membership status of each node is recorded in the membership vector [8]. The bit size of the membership vector **N** reflects total number of participating nodes in all TDMA rounds of a cluster cycle such as N={1,2,3,...,n}.

The membership service informs all the nodes about active and inactive nodes with a latency of one TDMA cycle [8]. The presented approach removes transmission slot overhead time in all TDMA rounds, therefore membership service latency is also improved as compared to traditional time-triggered protocols such as FlexRay. Similarly, no explicit membership information or acknowledgment is required, as the corresponding information can be derived from the embedded frame CRC calculation, further reducing overhead through this implicit acknowledgment mechanism.

Algorithm 1 represents the mechanism of membership service [8] as a transmitter node. It has a membership vectors with a size of total number of active nodes in the cluster. Please note, the term *active nodes* is used for those who are transmitting frames in their allocated transmission slot. The node who is passive such as an actuator node and has nothing to transmit should not be the part of membership vector. A transmitter sets its membership flag to TRUE in the membership vector before calculating the CRC on the CState (Controller State). The CRC value is embedded in the frame, therefore, the CState is transmitted implicitly, saving more bandwidth on the network [8]. The term *remoteCRC* is used to represent the value of CRC calculated on its controller state and same is true for every transmitter node. When the local clock of a transmitter node reaches an instant that is marked as the transmission time for the node, the node starts transmitting the frame over the communication channel.

Algorithm 1 Membership service for a Transmitter node

```
Require: $sizeof(MembshipVector) = n \in N$
1: Let $T_{i,j}$ is a transmitter node in slot i of TDMA round j
2: if ($currentTime == timetoTransmit$) then
3:     set flag of $T_{i,j}$ in membership vector as TRUE
4:     set $agreedSlotCounter$ to ONE
5:     $remoteCRC \leftarrow PerformCRConCState$
6:     Transmit Frame
7: else
8:     Wait to transmit
9:     Go to 2
10: end if
```

Algorithm 2 represents the mechanism of membership service [8] as a receiver node. Before receiving the frame from a transmitter say $T_{i,j}$, it sets the membership flag of $T_{i,j}$ as TRUE in its membership vector. Then it calculates the CRC check on its controller state. The calculated CRC value is represent by the term *localCRC* and same term is used for all the receiver nodes. When the local clock reaches an instant that is marked as the time to receive a frame from $T_{i,j}$ in the MEDL then it starts receiving. The CRC value it receives as *remoteCRC* (processed by the transmitter node) will be compared with *localCRC* (processed by the receiver node) and if the result is true then it means the frame received from $T_{i,j}$ is intact, otherwise it will set the membership flag to FALSE for $T_{i,j}$ in its membership vector. A CRC check will be failed in-case of a mode change request. As the mode change is not deterministic and requested at run time therefore transmitter and receiver nodes' controller state will disagree, which results in a CRC failure. To avoid the complexity, the proposed approach allowed only selected number of nodes those can make a mode change request. Therefore, when the CRC check fails, the receiver node will check whether transmitter node is authorized to initiate the mode change request and if so, then receiver node will update its CState and re-compute the CRC value. Then the receiver will compare *localCRC* with *remoteCRC* and sets the membership flag accordingly, based on the result of CRC comparison, i.e., either true or false.

Algorithm 2 Membership service for a Receiver node

```
Require: $T_{i,j}$ as transmitter node
 1: Let $R_{i,j}$ is a receiver in slot i of TDMA round j
 2: $R_{i,j}$ sets membership flag of $T_{i,j}$ as TRUE in its membership vector
 3: $localCRC \leftarrow PerformCRConCState$
 4: if ($currentTime == timeToReceiveFrame$) then
 5:     Receive frame
 6: else
 7:     wait to receive
 8:     Go to 4
 9: end if
10: Compare CRC values
11: if ($remoteCRC == localCRC$) then
12:     $R_{i,j}$ received correct frame from $T_{i,j}$
13: else
14:     If $T_{i,j}$ is authorized to perform mode change
15:     $R_{i,j}$ update it CState and recompute $localCRC$
16:     in-case result of ($remoteCRC == localCRC$)
17:     $R_{i,j}$ received correct frame from $T_{i,j}$
18: else
19:     $R_{i,j}$ sets membership flag of $T_{i,j}$ as FALSE in its membership vector
20: end if
```

Frame Acknowledgment: The acknowledgment of a frame happens implicitly through the membership service, by including the CState into each payload frame's CRC calculations [33]. A transmitter transmits replicated frames on two different communication channels, and, if any one of them is received correctly by a receiver then it will consider the transmitter as an active node at its *membership point* (a post receive phase after the transmission phase of a transmitting node).

If a transmitting node views itself as fully functional then it sets its own membership flag to TRUE in its membership vector and its *agreedSlotCounter* to one as shown in Algorithm 1. If the successor ($T_{i+1,j}$) of the transmitting node $T_{i,j}$ has received a correct frame on any of the two replicated communication channels then membership flag of $T_{i,j}$ is set to TRUE in the membership vector of $T_{i+1,j}$, therefore, $T_{i,j}$ can use $T_{i+1,j}$ transmission as an acknowledgment. The $T_{i,j}$ will only consider the transmission of $T_{i+1,j}$ as an acknowledgment if it receives a correct frame on any of the replicated communication channels. Otherwise, the membership flag of $T_{i+1,j}$ is set to FALSE in the membership vector of $T_{i,j}$ and $T_{i,j}$ will look for the transmission of $T_{i+2,j}$ (next successor) to find an acknowledgment of its transmission.

When $T_{i,j}$ acts as a receiver to receive a frame from its successor, say $T_{i+1,j}$, then there are two cases that are checked when performing CRC calculations on the received frame.

- **CASE 1:** $T_{i,j}$ sets its own and $T_{i+1,j}$ membership flag to TRUE in its membership vector and then performs CRC calculations on its local CState. Then a comparison is performed on $localCRC$ and $remoteCRC$.

- **CASE 2:** $T_{i,j}$ sets its own membership flag to FALSE and $T_{i+1,j}$ membership flag to TRUE in its membership vector and then performs CRC calculations on its local CState. Then a comparison is performed on $localCRC$ and $remoteCRC$.

If the result of CASE 1 is TRUE then $T_{i,j}$ assumes that its transmission was correct and remains in the membership vector. The $T_{i,j}$ will also increased its $agreedSlotCounter$ by one. But if the result of CASE 1 is FALSE then CASE 2 will be considered and if the result of CASE 2 is TRUE then it means either transmission of $T_{i,j}$ was not successful or $T_{i+1,j}$ made some error. At this stage, it is not confirmed whether $T_{i,j}$ or $T_{i+1,j}$ is correct, therefore, $T_{i,j}$ will look for its second successor i.e. $T_{i+2,j}$. If CASE 1 and CASE 2 both fail then it can be predicted that either transmission of $T_{i+1,j}$ is corrupted or $T_{i+1,j}$ is not operational at all. If transmission activity from $T_{i+1,j}$ on any of the replicated communication channel is observed then $T_{i+1,j}$ will be considered as faulty and $failedSlotCounter$ will be incremented by one. If $T_{i,j}$ is unable to make a decision by using CASE 1 and CASE 2 then it will use the transmission of its second successor i.e. $T_{i+2,j}$ and following two cases will be tested.

- **CASE 3:** $T_{i,j}$ sets its own and $T_{i+2,j}$ membership flags to TRUE while $T_{i+1,j}$ membership flag to FALSE in its membership vector and then performs CRC calculations on its local CState. Then a comparison is performed on $localCRC$ and $remoteCRC$.

- **CASE 4:** $T_{i,j}$ sets its own membership flag to FLASE while $T_{i+1,j}$ and T_{i+2j} membership flags to TRUE in its membership vector and then performs CRC calculations on its local CState. Then a comparison is performed on $localCRC$ and $remoteCRC$.

The result of CASE 3 (if TRUE) indicates that transmission of $T_{i,j}$ was correct and $T_{i+1,j}$ was faulty. Therefore, $T_{i+1,j}$ will be removed while $T_{i,j}$ will remain in the membership vector. Both $agreedSlotCounter$ and $failedSlotCounter$ will be incremented by one. Thus, $T_{i,j}$ is acknowledged. Otherwise, if the result of CASE 4 is TRUE then it means original transmission from $T_{i,j}$ was erroneous and transmission from $T_{i+1,j}$ was correct. Therefore, $T_{i,j}$ will remove it from membership vector. Both $agreedSlotCounter$ and $failedSlotCounter$ will be incremented and $T_{i,j}$ marks the transmission of $T_{i+2,j}$ as correct. $T_{i,j}$ will consider its transmission as not acknowledged and if acknowledgment failure reaches to its maximum value (defined in MEDL) then $T_{i,j}$ will freeze its controller. In worst case, if both CASE 3 and CASE 4 fails then $T_{i+2,j}$ will be removed from membership vector and $failedSlotCounter$ will be incremented by one. $T_{i,j}$ will choose $T_{i+3,j}$ as second successor and the loop continues depending upon the number of nodes in the TDMA round.

4.3.2.2 Clique avoidance

It is possible that an erroneous condition may leave a cluster with multiple cliques where a few nodes do not agree with the rest of the nodes on their CState and hence, remove them from their membership list [8]. This leads to formation of multiple cliques [8]. To avoid such errors, a clique avoidance algorithm [141] is implemented as shown in Algorithm 3. A point in time, when a node reaches a conclusion about its CState agreement with the rest of the nodes in the ensemble, is called a membership recognition point [8]. Different nodes may reach this point at different points in time. At its membership recognition point, a node is able to decide whether majority of the nodes are in agreement with its CState by using two of its counters [8]. These are $agreedSlotCounter$ (shows the number of other nodes in a TDMA round that are agree with the slot status of the node) and the $failedSlotCounter$ (shows the number of other nodes in a TDMA round that do not agree with the slot status of the node). If the node in question resides within a majority clique, then it will continue its functioning, otherwise it has to restart and reintegrate into the cluster [141].

Algorithm 3 Clique avoidance

```
Require: with latency of one TDMA round and before transmitting the frame
    in next TDMA round
1: if $agreedSlotCounter > failedSlotCounter$ then
2:     agrees with majority of cluster nodes
3:     set $0 \leftarrow agreedSlotCounter$
4:     set $0 \leftarrow failedSlotCounter$
5:     do transmit the frame
6: else
7:     freeze the CC
8:     restart in healthy state and reintegrate with the cluster
9: end if
```

4.3.2.3 Clock Synchronisation

To enable the deterministic behaviour of the protocol so that each node knows the precise point of its pre-defined schedule saved in its local MEDL for sending and receiving information over the shared channel, it is essential for each node to synchronise its clock with the other nodes in order to establish a global time base and run the protocol smoothly [68]. Clock synchronisation in INCUS+ follows an analogous procedure to that discussed in INCUS and [33]. The key difference here, in contrast to these time-triggered protocols, is the resynchronisation point $R_p(t)$ – a slot defined in the MEDL to perform clock synchronisation. Let k be the number of slots set as resynchronisation point then R_p duration becomes:

$$R_p^{ttp}(t) = k * \tau^{max} \tag{4.15}$$

where t be the real time and τ^{max} is the slot length. In INCUS+, the formula needs to be adjusted to accommodate different slot lengths:

$$R_p^{incus+}(t) = \sum_{i=0}^{k-1} S_i(t) \tag{4.16}$$

where $S_i(t)$ represents slot length and $(k \geq 4)$.

The resynchronisation interval in INCUS+ is less than the resynchronisation interval of traditional protocols such as Flexray and TTP. The reason for this is the node slot net idle time, which is zero for all node slots in the my approach. This provides the ground for fast-tracking the process of clock synchronisation. The fault-tolerant average algorithm [59] which is formally verified in [142] to synchronise the clocks of all nodes in the cluster is shown in Algorithm 4.

4.3.3 Autonomous vehicle case study

We shall now analyse the efficiency gains in a representative case study. The bandwidth requirements to deploy ADAS systems are very high, specifically by adding multiple cameras with different focal lengths to detect multiple objects such as traffic lights, pedestrians, road signs, etc [135]. Therefore, multiple compression techniques are used to handle huge data traffic in in-vehicle networks [135]. A generic video compression technique, H.264, has been used in this case study and it has been shown by Tankred Hase *et al.* [143] that using the H.264 compression at a low bit rate of up to 0.125Mbps provides sufficient quality to detect an obstacle. To keep it simple, the need of flexible TDMA round lengths, i.e., the length of rounds may vary depending on differing payloads of a node in different TDMA

Algorithm 4 Clock Synchronisation

```
Require: received correct frame only
 1: Initialised stack of four with zero
 2: Capture the time interval when received the first bit from sending node
 3: Let $\hat{t_{RF}}$ is the actual time when receiver starts receiving the frame from the
    sender
 4: Let t$_{RF}$ is time when receiver supposed to receive the frame from the sender
    - defined in MEDL
 5: Calculate correction term $\Delta_{corrterm}$
 6: $\Delta_{corrterm} = \hat{t_{RF}} - t_{RF}$
 7: PUSH $\Delta_{corrterm}$
 8: if $R_p^{incus+}(t)$ then
 9:     POP the values
10:     discard the largest and smallest values
11:     take the average of remaining two values
12:     apply this average to correct the local clock
13: else
14:     wait for the interval $R_p^{incus+}(t)$
15:     push new values of $\Delta_{corrterm}$ to the stack
16: end if
```

rounds is elaborated through this case study with single mode of operation. The autonomous vehicle moves along the road and avoiding obstacles as detected by sensors such as cameras. Importantly, emergency braking systems depend on these sensors to detect critical situations, such as a pedestrian on the road, that require automated emergency braking within a tightly constraint deadline to prevent a collision.

To develop a safety-critical, distributed real-time system node cluster for this case study, the concept of a platooning system [144][9] and a pedestrian detection system [145] is utilised.

The layout of the autonomous vehicle is shown in Figure 4.9. The basic building blocks of the system are the Sensor fusion Node (SFN), Vehicle Controller Node (VCN), and a Camera with on-board with H.264 compression. Two redundant SFNs are deployed to improve fault tolerance in the hardware domain. The components, VCN and Camera, are not replicated as the drive controller interface is able to perform a fail-safe operation (emergency stop) if it does not get a valid life sign from the VCN. The communication among camera, SFN and VCN is

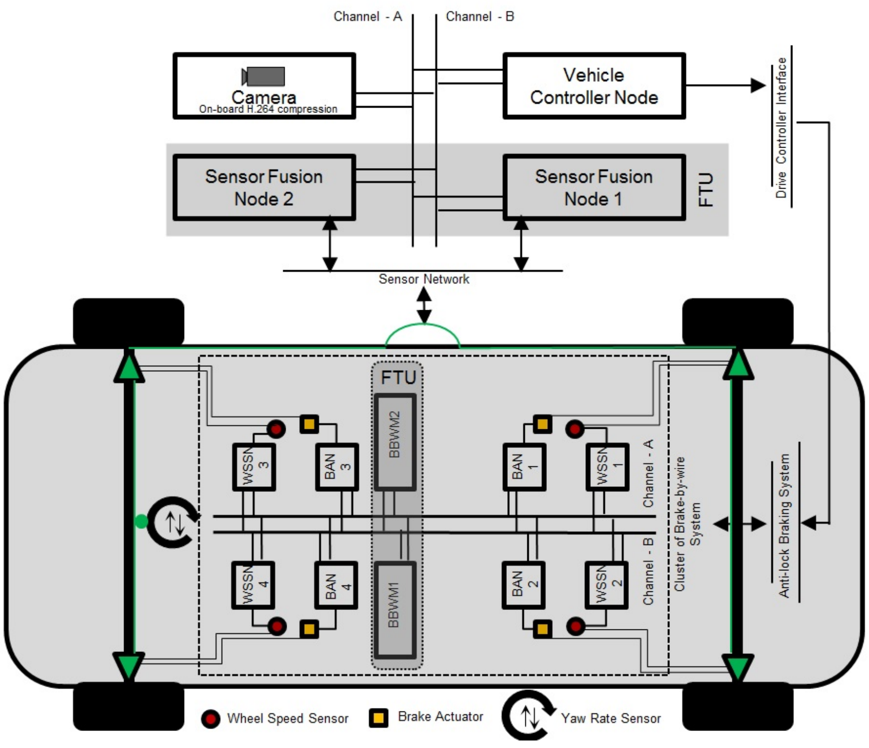

FIGURE 4.9: Layout of Autonomous Vehicle

performed through replicated communication channels with a communication capacity of 1Mbps. The layout of a time-triggered cluster for the case study is shown in Figure 4.10.

Each node in the cluster is divided into three layers, Host, Communication Network Interface (CNI), and Communication Controller (CC) layer as shown in Figure 4.11. The CC is the part of the communication subsystem and it interacts with the host through the CNI. To avoid the increased design complexities of the protocol, Logic-Labelled Finite State Machines (LLFSMs) are used to implement the functionality of each communication controller [146]. A subsumption architecture is used, where the functionality of each controller is decomposed into sub-machines. The host processes the information from/to sensors/actuators and exchanges the data with the CC through the CNI. The CC fetches and transmits the data from and to the communication channel at the periodic instances stored in the MEDL. Each CC has the copy of the MEDL and, hence, knows in advance when to transmit and receive the data to and from the communication channel. This mechanism guarantees the deterministic behaviour of the protocol.

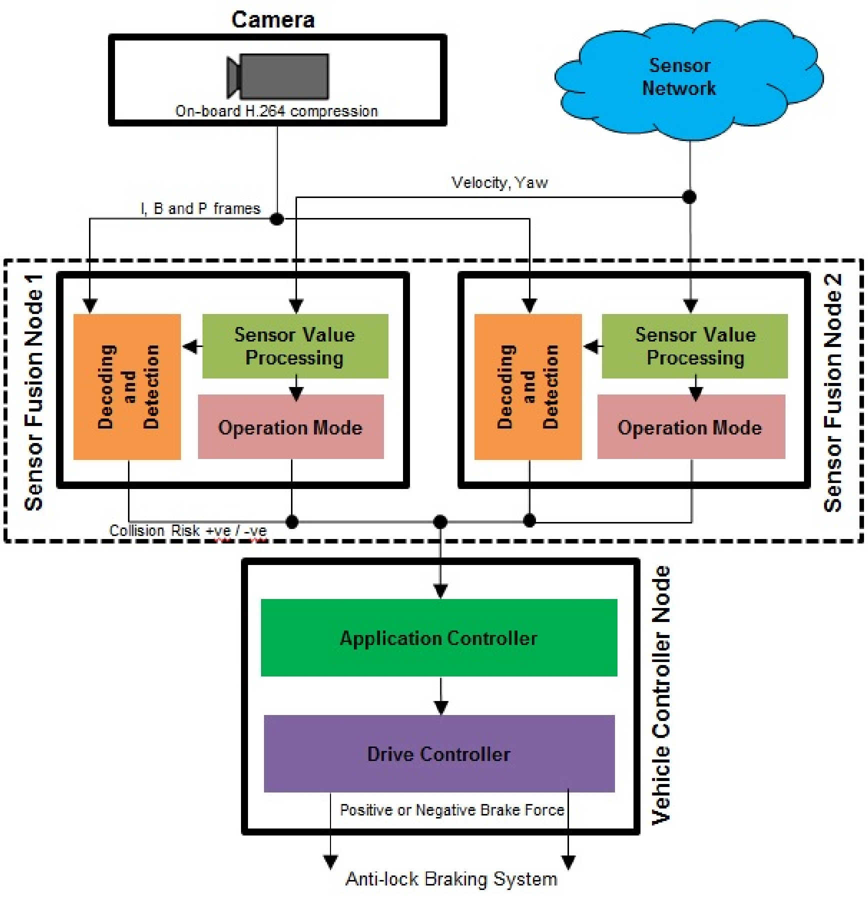

FIGURE 4.10: Layout of Time-Triggered Class-C Network of Autonomous Vehicle [9]

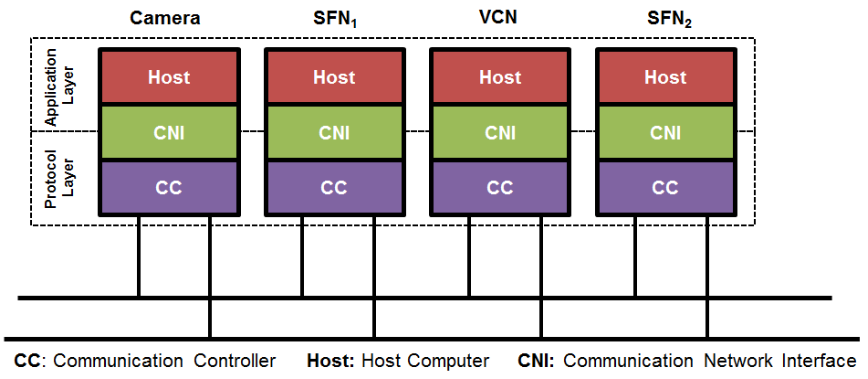

FIGURE 4.11: Internal structure of each node in the protocol cluster

4.3.3.1 Camera

The camera is transmitting compressed frames such as Intra-coded (I), Bidirectional (B), and Predicted (P), frames by using the H.264/AVC codec. To deliver videos in real-time, a frame from a camera (e.g. an I-frame) is transmitted in a

TDMA round, followed by the next frame (e.g. B-frame) in the next round, and so on. These frames are of significantly different sizes: an I-frame is many times larger than P and B-frames. Using the same transmission slot length (required by the I-frame) in the following TDMA rounds to send B and P-frames results in huge inefficiency regarding channel utilisation as payloads of B and P-frames are much smaller than I-frames. and a GOPs (Group of Pictures) configuration is used with M = 2 (distance between two anchor frames) and N = 3 (distance between two full images) which leads to the sequence `IBPIBP` as shown in Figure 4.12. The camera is transmitting GOPs on the basis of one frame per round, therefore a complete sequence of the compressed frames is transmitted in three different TDMA rounds. Consequently, the cluster cycle of the case study consists of three TDMA rounds.

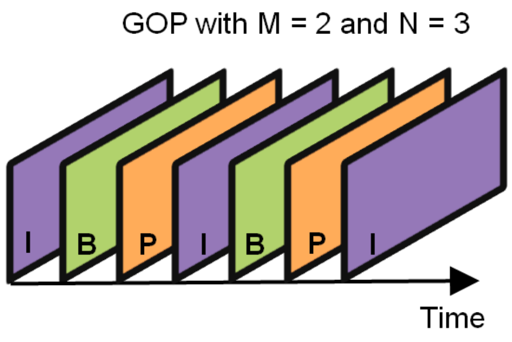

FIGURE 4.12: GOP configuration

As the camera is transmitting different size frames in different TDMA rounds of a cluster cycle, the slot length of this node should be different for each TDMA round (Table 4.4).

4.3.3.2 Sensor Fusion Node

The SFN decodes the encoded stream received from the camera and forwards it to the detection module. The principle of operation to detect a pedestrian is same as used in [145]. The detection module extracts the region of interest in each frame and to detect a pedestrian, it exploits the fact that object size increases while relative distance decreases [145]. The distance to a possible collision is calculated on the basis of change in object size, velocity, and yaw of the autonomous vehicle.

Table 4.4: The Ideal Transmission slot length for each Node of the Autonomous Vehicle System during a Cluster Cycle

First TDMA Round			
Node Name	Data	Frame Length	Slot Length
Cam-I	75032 bits	$75032 + 28 = 75060$ bits	75060 μs
SFN	8 bits	$8 + 28 = 36$ bits	36μs
VCN	Nil	$0 + 8 = 8$ bits	8μs
Second TDMA Round			
Node Name	Data	Frame Length	Slot Length
Cam-B	4400 bits	$4400 + 28 = 4428$ bits	4428μs
SFN	8 bits	$8 + 28 = 36$ bits	36μs
VCN	Nil	$0 + 8 = 8$ bits	8μs
Third TDMA Round			
Node Name	Data	Frame Length	Slot Length
Cam-P	20392 bits	$20392 + 28 = 20420$ bits	20420μs
SFN	8 bits	$8 + 28 = 36$ bits	36μs
VCN	Nil	$0 + 8 = 8$ bits	8μs

This follows the detailed functionality of pedestrian detection found in [145]. The sensor value processing module takes velocity and yaw rate value from the sensor network and forwards it to the detection module [145]. The decoding and detection module sends an output of 8 bits to the VCN that informs the actuator node about a possible risk of collision and the VCN acts accordingly [145]. To keep it simple, a single mode of operation is used in this case study as discussed above and the state of each node is marked as a ready state.

4.3.3.3 Vehicle Controller Node

The Application controller in the VCN receives the information on collision risk and passes on this information to the Anti-lock-braking System (ABS) through its drive controller interface. All modern vehicles are equipped with the ABS system as it helps to stop the vehicle on slippery roads by avoiding uncontrolled skidding. The cluster of the ABS system consists of Wheel Speed Sensors Nodes (WSSN), Brake Actuator Nodes (BAN) and Brake-by-Wire Manager Nodes (BBWM). ABS sits on top of the Brake-by-wire system and is responsible for applying automated brakes to stop the vehicle when it receives a positive signal for brake force from the VCN. A detailed description of the ABS's cluster can be found in the Section 4.2.1.

For the case study, the VCN is designed to send an initialisation-frame in each TDMA round. This is because the VCN acts as an actuator node and does not need to transmit any application data. This approach helps to speed up the reintegration process of recovering nodes.

4.3.4 Performance Comparison

In this section, the logical computations are used to analyse the impact of fixed and equal length slot configurations on bandwidth and channel utilisation. We will explore the computed results of INCUS+ and compare them with an existing TTA-based communication protocol, FlexRay. Table 4.5 shows a descriptive list of the terms used in the computational model in order to compare FlexRay and INCUS+. Please note, for simplicity, the proposed approach is not using the timing parameters discussed in the Section 6.3 as these, while hardware-dependent, would be the same for all approaches.

4.3.4.1 FlexRay slot allocation

τ^{max} of each node comprises of T_i^{trans} and T_i^{idle} as given in Figure 4.13. T_i^{ovhd} and T_i^{idle} of $node_i$ are:

$$T_i^{idle} = \tau^{max} - T_i^{trans} \tag{4.17}$$

$$T_i^{ovhd} = T_i^{idle} + T^{ifg} \tag{4.18}$$

$\hat{T}^{ovhd\_r}$ for $\boldsymbol{n}$ number of nodes in a TDMA cycle should be:

$$\hat{T}^{ovhd\_r} = \sum_{i=0}^{n-1} T_i^{ovhd} \tag{4.19}$$

TABLE 4.5: Definition of the terms used in the computational model

Symbol	**Description**
T^{fr_r}	Length of FlexRay TDMA round (where $T^{fr\_r_m} = T^{fr\_r_n}$)
T^{inc+_r}	Duration of TDMA round in INCUS+ (where $T^{inc+\_r_m}$!= $T^{inc+\_r_n}$ is possible)
τ^{max}	Slot length of each node in FlexRay approach (where $\tau_i^{max\_r_m} = \tau_j^{max\_r_m}$ and $\tau_i^{max\_r_m} = \tau_i^{max\_r_n}$)
τ^{inc+}	Slot length of each node in INCUS+ (where $\tau_i^{inc+\_r_m}$!= $\tau_j^{inc+\_r_m}$ and $\tau_i^{inc+\_r_m}$!= $\tau_i^{inc+\_r_n}$ is possible)
T_i^{trans}	Transmission time for control information and application data for node i during its allocated node slot in TDMA round
T_i^{idle}	Node slot idle time for node i that is not utilised to transmit application data or control information
T^{ifg}	Time when there is no transmission between frames known as Inter Frame Gap (IFG) overhead time. This is used to accommodate latency and jitter because of propagation delay and clock synchronisation limits etc.
T_i^{ovhd}	Total overhead for node i for its allocated slot
$\hat{T}^{ovhd\_r}$	Total overhead time in a TDMA round
$\hat{T}^{ovhd\_c\_cycle}$	Total overhead time in a cluster cycle
$T^{fr\_c\_cycle}$	Total length of cluster cycle in TTA based slot allocation approach
$T^{inc+\_c\_cycle}$	Total length of cluster cycle in INCUS+ approach

If there are $\boldsymbol{k}$ number of TDMA rounds then the total overhead time ($\hat{T}^{ovhd\_c\_cycle}$) in a cluster cycle can be calculates as:

$$\hat{T}^{ovhd\_c\_cycle} = \sum_{i=1}^{k} \hat{T}_i^{ovhd\_r} \tag{4.20}$$

T^{fr_r} for $\boldsymbol{n}$ number of nodes is:

$$T^{fr\_r} = n \cdot (\tau^{max} + T^{ifg}) \tag{4.21}$$

The length of a $T^{fr\_c\_cycle}$ in FlexRay slot allocation approach with $\boldsymbol{k}$ number of TDMA rounds should be:

$$T^{fr\_c\_cycle} = k \cdot (T^{fr\_r}) \tag{4.22}$$

By using Equation (4.20) and Equation (4.22), CU in a FlexRay cluster cycle can be calculated as:

$$CU = \left[\frac{T^{fr\_c\_cycle} - \hat{T}^{ovhd\_c\_cycle}}{T^{fr\_c\_cycle}} \right] .100 \qquad (4.23)$$

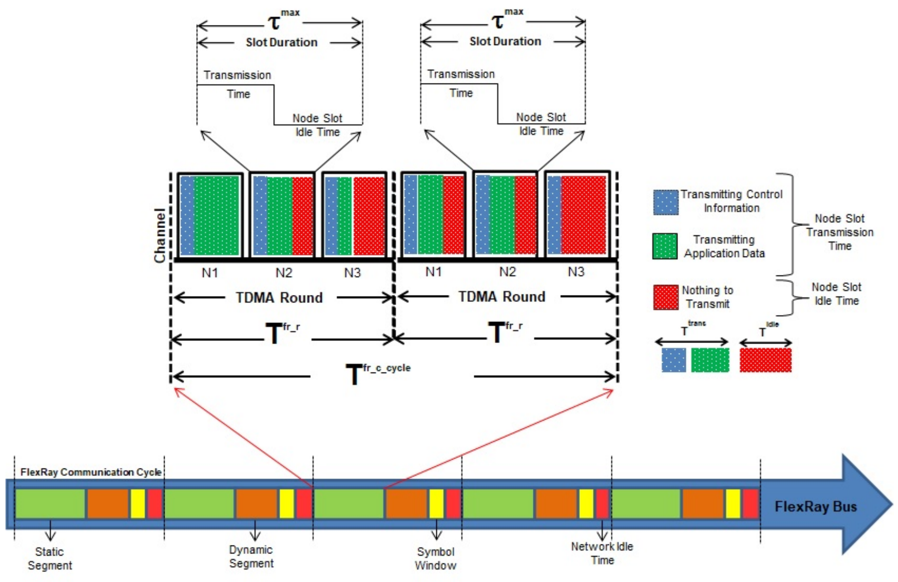

FIGURE 4.13: FlexRay slot allocation over a Cluster Cycle

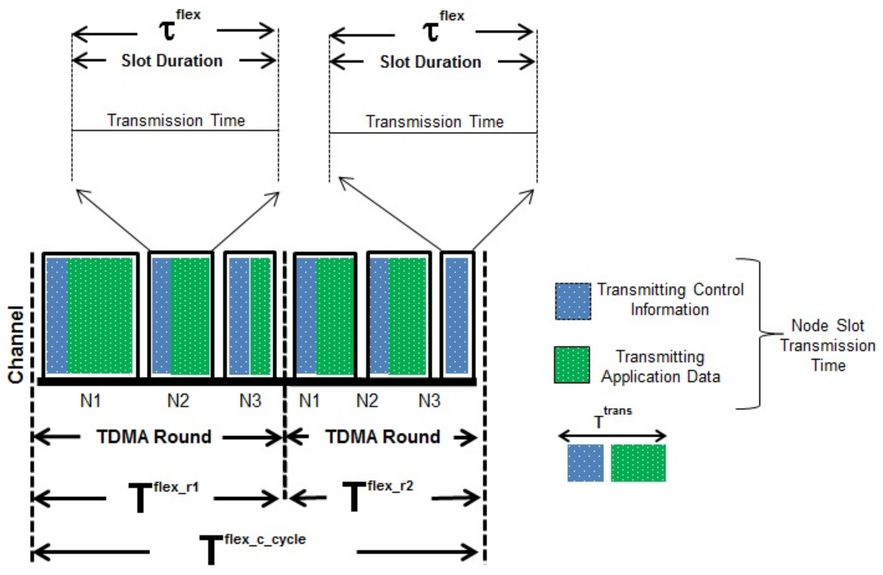

FIGURE 4.14: INCUS+ slot allocation over a Cluster Cycle

4.3.4.2 INCUS+ slot allocation

Let us now evaluate the proposed, INCUS+ approach, where the length of the slot for each node is customised in relation to its transmission payload in each

TDMA round of the cluster cycle. This customisation avoids the overhead time as demonstrated in Figure 4.14. Therefore, T_i^{idle} is zero and τ_i^{inc+} in each TDMA round becomes:

$$\tau_i^{inc+} = T_i^{trans} \tag{4.24}$$

For a number of nodes $\boldsymbol{n}$, T^{inc+_r} in INCUS+ is:

$$T^{inc+\_r} = \sum_{i=0}^{n-1} (\tau_i^{inc+} + T^{ifg}) \tag{4.25}$$

Similarly, for a number of TDMA rounds $\boldsymbol{k}$, $T^{inc+\_c\_cycle}$ becomes:

$$T^{inc+\_c\_cycle} = \sum_{i=1}^{k} (T_i^{inc+\_r}) \tag{4.26}$$

CU in a cluster cycle then can be defined by using Equation (4.20) and Equation (4.26):

$$CU = \left[\frac{T^{inc+\_c\_cycle} - \hat{T}^{ovhd\_c\_cycle}}{T^{inc+\_c\_cycle}} \right] .100 \tag{4.27}$$

As it is now guaranteed that $T_i^{ovhd} = 0$ for every i, every node will fully utilise its allocated τ_i^{inc+} for transmitting application data and control information over the channels. This results in transmitting the same data while removing the slot overhead of previous approaches. The INCUS+ slot allocation approach therefore avoids the additional overhead caused by unequal transmission payloads across TDMA cycles.

Let us now analyse how flexibility improves the performance of the protocol by illustrating its impact on the autonomous vehicle case study.

4.3.4.3 Impact of flexibility on overhead time

Every node in the autonomous vehicle system has a unique functionality and, hence, requires a different transmission time. The slot length of a node not only differs from other nodes, but also the same node may need different transmission

Table 4.6: Allocated slot length and potential overhead time in FlexRay slot allocation approach

First TDMA Round		
Node	τ^{max}	T_i^{ovhd}
Camera	75060μs	$(0+4) = 4\mu$s
SFN$_{1,2}$	75060μs each	$(75024+4).2 = 150056\mu$s
VCN	75060μs	$(75032+4) = 75036\mu$s
Second TDMA Round		
Node	τ^{max}	T_i^{ovhd}
Camera	75060μs	$(0+4) = 4\mu$s
SFN$_{1,2}$	75060μs each	$(75024+4).2 = 150056\mu$s
VCN	75060μs	$(75032+4) = 75036\mu$s
Third TDMA Round		
Node	τ^{max}	T_i^{ovhd}
Camera	75060μs	$(0+4) = 4\mu$s
SFN$_{1,2}$	75060μs each	$(75024+4).2 = 150056\mu$s
VCN	75060μs	$(75032+4) = 75036\mu$s

requirements in different TDMA rounds, such as the camera in this case study. For the FlexRay approach, τ^{max} for each node would have been 75060 microseconds, as required by the camera to transmit maximum size packets (camera I-frames). Therefore, the FlexRay slot allocation approach adds a significant T^{idle} to each node slot that requires less transmission time than assigned, as shown in Figure 4.13. Hence, FlexRay would get a significant transmission overhead time for most slots.

A much better result can be achieved using my flexible slot allocation approach, as the transmission times of all τ^{inc+} are configured on the basis of their actual transmission requirements in each TDMA round of a cluster cycle. Here, the camera is allocated different transmission slot lengths of 75060 microseconds, 4428 microseconds, and 20420 microseconds for the first, second, and third TDMA rounds respectively. While still statically configured, this slot allocation approach is more flexible and, importantly, eliminates the T^{idle} in each slot as shown in Figure 4.14. Therefore, T^{idle}=0 for all node slots over the cluster cycle, which significantly reduces the T_i^{ovhd} for a $node_i$.

Why is fragmentation not feasible? One might wonder, what if taking the same fixed and equal-length slot allocation approach as demonstrated by FlexRay and TTP, but configure it with zero slot idle time by allocating to all nodes the minimum (instead of maximum) slot length required by a node in the whole cluster? As per the case study, VCN is the node that requires 28 microseconds slot length to transmit an initialisation frame. This slot length can be configured for each node in the cluster which eliminates slot idle time. However, this may need fragmentation for the nodes having bigger frames such as Camera and SFN nodes. A fragmentation of a larger frame can be used to accommodate the frame within the duration of transmission slot length and this single frame can be transmitted over multiple TDMA rounds. Each camera transmits three frames with different payload length such as I, B and P-frames as discussed above. Let us have a look how many slots are required to transmit an I-frame using this approach. This will need 2680 slots to transmit an I-frame by the camera node. This means, instead of one TDMA round, 2680 TDMA rounds are required to transmit a complete I-frame when using fragmentation. Each fragment of an I-frame is transmitted once in a TDMA round to avoid the complexity such as giving more weightage to one node by allowing it to transmit more than once in a TDMA round as compared to other nodes in the membership service. If such node is actually a faulty node or a single fragmentation becomes faulty during transmission, then this can disrupt the whole fault tolerance mechanism ensured by the membership service. An extra overhead of 75040 microseconds will be incurred through appended control information in each frame, while an extra 10720 microseconds overhead time is required for inter-frame gap (IFG) time as compared to my proposed approach, which requires 28 microseconds of control information and 4 microseconds for IFG to transmit an I-frame. Similarly, if B and P-frames are considered using multiple fragments as well as multiple fragments for the SFN node, then this transmission overhead time will be huge, and results in significant transmission latency. Moreover, my approach decouples the time to detect an error, by using the membership service, which is independent of the frame payload. For example, if one or more fragments failed to reach the destination due to any error on the network, then the whole message from the transmitter node would have to be discarded. This fragmentation approach therefore would add significant overhead and system complexity, as it requires additional error handling above the protocol level, i.e. it would need additional higher level services for fault tolerance. Those higher-level services would need to interact with the protocol-level fault tolerance mechanism, creating unnecessary coupling and thus, impeding on decomposability.

Allowing a variable slot length for each node on the basis of its payload requirements may, at first glance, appear to result in jitter, caused by data transmission at irregular intervals (due to the unequal TDMA slot lengths). However, this variability is deterministic and known in advance, as all communication schedules are created at design time. I therefore argue that this does not actually constitute jitter (which would cause inherent uncertainties due to its stochastic nature), as here, the exact latency is known in advance. As long as the transmitted data are phase-insensitive, taking into account a simple measure such as the maximum latency is sufficient to allow the system to be designed in the same fashion as with existing TTA-based protocols. If, on the other hand, the transmitted data are phase-sensitive, the exact interval at which data are scheduled to be transmitted (i.e. the timing of the corresponding TDMA slot as designed) needs to be taken into account when interpreting the data. It should be noted that in either case, the maximum latency of INCUS+ is expected to be better (and guaranteed to be no worse) than that of TTP/C or FlexRay.

4.3.4.4 Impact of flexibility on Channel Utilisation

We will now analyse how INCUS+ can improve channel utilisation in our scenario. Four nodes are connected via the replicated bus[2] (see Figure 4.10). In a FlexRay approach, τ^{max} for each node in all TDMA rounds would be 75060μs. Only one node, the camera (during first TDMA round) makes full use of τ^{max} for transmission and all remaining slots, including those of the camera in subsequent TDMA rounds, will have T^{ovhd} in their allocated τ^{max} (see Table 4.6, Figure 4.15). By using four bits time (4 μs) for T^{ifg}; slot length, plus T^{ifg}, is 75064 μs for each

[2]The timing requirements of both replicated buses are exactly the same.

TABLE 4.7: Allocated slot length and potential overhead time in INCUS+ slot allocation approach

First TDMA Round		
Node	τ^{inc+}	T_i^{ovhd}
Camera	75060μs	$(0+4) = 4\mu$s
SFN$_{1,2}$	36μs each	$(0+4)\cdot 2 = 8\mu$s
VCN	28μs	$(0+4) = 4\mu$s
Second TDMA Round		
Node	τ^{inc+}	T_i^{ovhd}
Camera	4428μs	$(0+4) = 4\mu$s
SFN$_{1,2}$	36μs each	$(0+4)\cdot 2 = 8\mu$s
VCN	28μs	$(0+4) = 4\mu$s
Third TDMA Round		
Node	τ^{inc+}	T_i^{ovhd}
Camera	20420μs	$(0+4) = 4\mu$s
SFN$_{1,2}$	36μs each	$(0+4)\cdot 2 = 8\mu$s
VCN	28μs	$(0+4) = 4\mu$s

node. Each slot is configured with 4 microseconds of Inter-Frame Gap (IFG) time. This accommodates the latency and jitter due to a number of factors, such as propagation delay and clock synchronisation limits. The same IFG time is set for all the existing protocols. Therefore, we can ignore latency and jitter impact here, as both, FlexRay and INCUS+ use exactly the same time limits to accommodate latency and jitter. According to Equation (4.21), the first T^{fr_r} is 300256μs where $\hat{T}^{ovhd\_r}$ by using Equation (4.19) is 225096 μs. Similarly, the values for T^{fr_r} and $\hat{T}^{ovhd\_r}$ in the second and third TDMA rounds are (300256μs, 295728μs) and (300256μs, 279736μs) respectively.

According to Equation (4.22), the length of a cluster cycle in the FlexRay slot allocation approach is 900768μs while the total overhead time according to Equation (4.20) is 800560μs. Substituting the above-mentioned values in Equation (4.23), channel utilisation for a cluster is only **11.124**% when following the FlexRay approach.

By comparison, in INCUS+, the τ^{inc+} of each node in each TDMA round is dependent on transmission requirements (see Table 4.7, Figure 4.16). Therefore the net overhead T^{ovhd} is zero for all nodes in each TDMA round. As T^{ifg} is 4μs

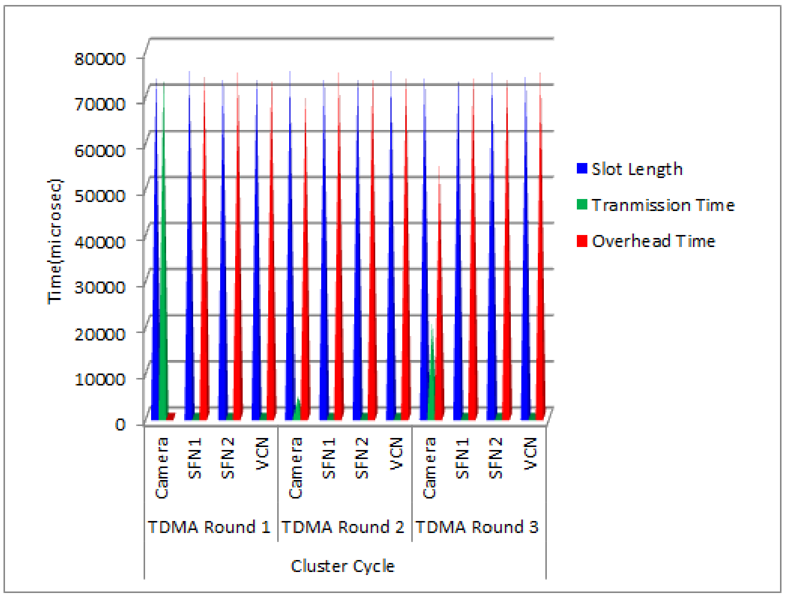

FIGURE 4.15: Slot length, transmission time and overhead time (where IFG = 4 bits time $\cong$ $\mathbf{4\mu}$s) using FlexRay slot allocation method.

and T^{idle} is zero, therefore, by using Equation (4.26) the length of a cluster cycle in INCUS+ is 100256μs while total overhead time according to Equation (4.20) is 48μs. Substituting the above-mentioned values in Equation (4.27) yields a channel utilisation for a cluster cycle in INCUS+ of **99.95**%. This is almost a ninefold increase in channel utilisation.

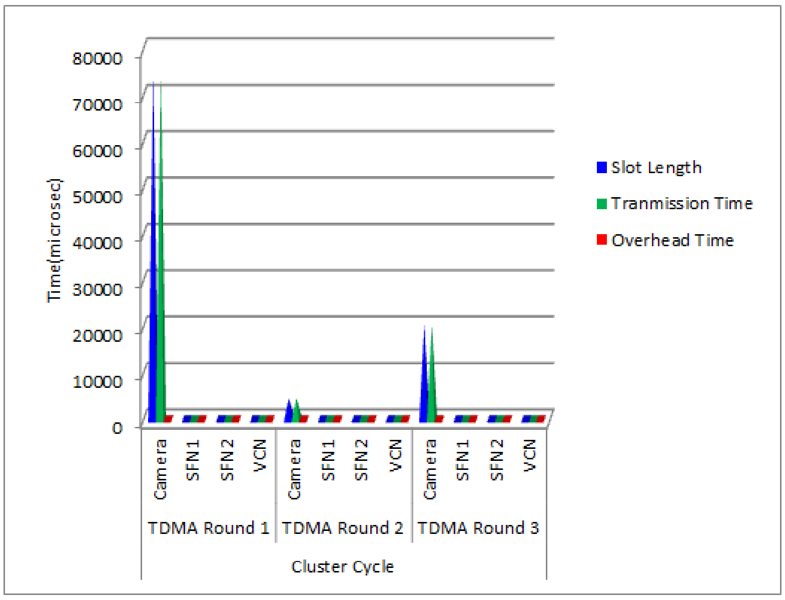

FIGURE 4.16: Slot length, transmission time and overhead time (where IFG = 4 bits time $\cong$ $\mathbf{4\mu}$s) using INCUS+ slot allocation method.

4.4 Flexible operational modes

The use-cases we have discussed so far demonstrate the need for flexible communication schedules not only inside a TDMA round but also, in different TDMA rounds. However, only a single, static configuration still exists for the system. A greater level of flexibility can be achieved by setting up different MEDLs (each with different communication schedule) for different operational modes. These MEDLs can then be switched at runtime. For example, a more complex use-case from the aviation industry where an on-board computer system in an aircraft controls different operational modes, e.g. based on whether the aircraft is on the ground or in the air [147]. To accommodate these different operational modes, a mechanism of mode changes was introduced in [147]. Mode changes are used to switch between statically defined but different operational modes [147]. In accordance with the changes in an operating environment, a controlling computer system performs mode changes, which means proceeding with different control patterns i.e. different task and message schedules [147]. Traditionally, the temporal control pattern does not change in a time-triggered (TT) systems [147], which currently restricts the achievable flexibility of TT real-time systems. There are a number of applications that have multiple distinct phases. For example, an aircraft flying from Brisbane to Sydney may have different operational phases such as taxiing on the airport, take off from the airport, climbing up in the air, level flight, descending in the air, landing at destination airport and then taxiing again [147]. However, different operational phases in [147] does not support flexible communication schedules. I have emphasis that each phase can be considered a different operating mode, which requires different control patterns. For example, a landing gear may need a different frame transmission frequency with different payload length during landing mode compared to level-flight mode with autopilot engaged. Consequently, it needs different length TDMA slots not only in different cluster cycles of an operating mode but also for different operational modes. Therefore, assigning same length transmission slots to a landing gear while in a landing mode as compared to level flight mode will result in poor channel utilisation. We will now examine how multiple operational modes, each with different communication schedules can be achieved. A comprehensive analysis of channel utilisation with different operational modes is not part of this section as it has already become evident from the previous case studies that a slot length configuration in accordance with the transmission payload will result in efficient bandwidth and channel utilisation. However, a formal verification of the impact of slot timings on system safety in different operational modes will be presented in Chapter 6.

4.4.1 Multiple operational modes

Safety-critical, distributed real-time systems can be configured to have different operational modes [147] at different points in time. Nodes operating as part of these systems exhibit different roles and therefore can have different transmission loads during each mode as illustrated in Figure 4.17. For example, a landing gear in an aeroplane transmits a different transmission payload during normal flight mode as compared to landing mode. Therefore the length of an allocated time-slot for a node can be different during different operational modes. Existing communication protocols for safety-critical systems are not capable of handling these different transmission schedules. They keep the same, fixed pattern of TDMA cycles and message lengths for all modes. There are vital reasons for this, as it was argued that bus guardians are not aware of any operational mode [147], so keeping dynamic TDMA pattern in different operational modes makes it difficult for guardian nodes to protect the network from babbling idiot faults. My proposed approach implements multiple operational modes with different communication schedules and lengths in each mode. The principle of operation is shown in this chapter, while the principles of bus guardians that can offer this level of flexibility are shown in Chapter 6.

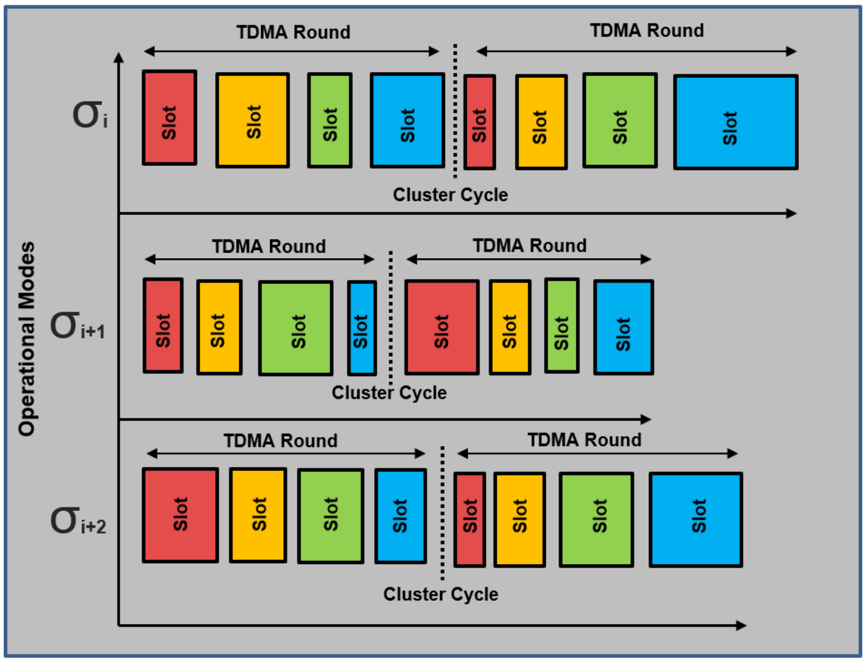

FIGURE 4.17: Slot length configurations during different operational modes

4.4.1.1 Implementation of multiple modes

The principle of operation of mode changes and mode handling in INCUS+ is the same as in TTP [33], except for supporting different communication schedules for each operational mode. Multiple MEDLs are initialised, each with a different communication schedule, as required by the respective operational mode. The protocol starts with a startup mode and after integration of all participating nodes in the ensemble, switches to one of the operational modes. To avoid a combinatorial explosion of the complexities, a mode change request is only initiated by a node that is allowed to do so and this request is embedded in the frame header. All the nodes in the ensemble that receive this request correctly will switch their mode either immediately or from the next TDMA cycle, depending upon the mode type configuration [147]. A mode change request contains a mode number that refers to the specific MEDL for that mode. An array that hold these references is called Mode Descriptor List (MODL) as shown in Figure 4.18. When acting upon the received mode change request, all nodes will replace their current MEDL with the new MEDL specified for the new mode. This new MEDL may contain a different communication schedule (with different message lengths) compared to the previous MEDL.

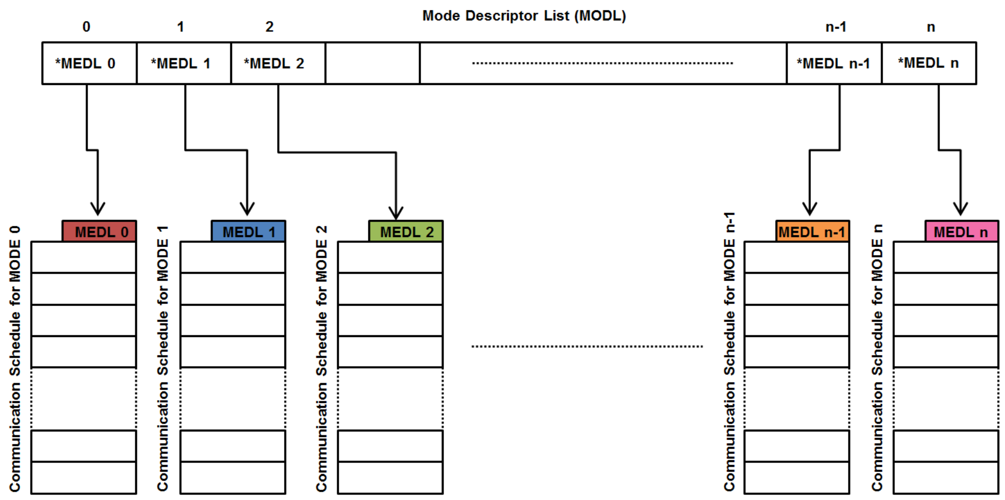
Mode Descriptor List (MODL)
0
1
2
n-1
n
*MEDL 0
*MEDL 1
*MEDL 2
*MEDL n-1
*MEDL n
MEDL 0
MEDL 1
MEDL 2
MEDL n-1
MEDL n
Communication Schedule for MODE 0
Communication Schedule for MODE 1
Communication Schedule for MODE 2
Communication Schedule for MODE n-1
Communication Schedule for MODE n

FIGURE 4.18: Mode Descriptor List

4.4.2 Mode handling

INCUS+ supports two types of mode change requests as discussed in [147], a deferred mode change and an immediate mode change. The difference between both modes is as follows:

4.4.2.1 Deferred Mode

Deferred mode changes can be used to reflect changes in normal operational conditions of the controlled object, for example, a change from taxiing mode to take-off mode in an aeroplane [147]. As stated above, deferred mode changes reflect the principle of consistency, i.e., maintaining an equal state among all nodes of a cluster and between the state of the protocol on the one hand and the state of the application on the other [147]. Deferred mode changes can be requested by an authorised node at any time during a cluster cycle. However, the change will only come into effect from the beginning of the next cluster cycle [147]. All tasks related to the previous mode of the participating nodes must be completed before the start of the new operational mode. This means that a system must be in ground state before the transition to a new mode [148].

A deferred mode change request during a cluster cycle remains pending until the start of the next cluster cycle [147]. All the ensemble nodes that successfully received the mode change request in the previous cluster cycle will make an autonomous transition to the new, requested mode at the start of next cluster cycle without any further notification. It is to be noted that a node which reintegrates into the ensemble only after the mode change request has been sent would have missed the mode-change request and thus will not be aware of this and hence, would not be able to adopt the new mode change from the next cluster cycle. This issue is resolved by making the mode change part of the controller state and therefore, a reintegrating node will also receive the pending mode change request whenever it receives the initialisation frame [147]. It is also possible that more than one request can be initiated during a cluster cycle from different authorised nodes. This issue can be resolved either by assigning static priorities to ensemble nodes or considering latest request as having the highest priority [147].

4.4.2.2 Immediate Mode

An immediate mode change can be requested by an authorised node at any time during a cluster cycle. Contrary to a deferred mode change, an immediate change applies right after the end of a slot in which the immediate mode change was requested [147]. The issues associated with a deferred mode change such as prioritising the mode change requests is not a problem with immediate mode as change to a new mode will take place immediately [147]. Similarly, this mode does not require to maintain information regarding pending mode change requests. Handling of immediate mode changes is easier than deferred mode change, as the system does not need to be in a ground state when immediate mode change applies [147]. Therefore, when an immediate mode change is invoked, all the tasks executing on the nodes in the ensemble must be aborted immediately [147]. However, this will not leave the system in an undefined state, as tasks pertaining to the new mode will commence.

4.5 Summary

Dependable safety-critical real-time (SCRT) communication protocols are becoming increasingly important and are being used in more complex applications. An example application of SCRT communication protocols we discussed is a self-driving car, where the protocol should have the ability to transfer a relatively large amount of safety-critical data (e.g. video) in real-time between various components of an autonomous vehicle.

The TTA approach for SCRT protocols has been to keep the protocol simple and inflexible, to achieve predictability and dependability. However, such an approach results in protocols that suffer from poor bandwidth and channel utilisation, particularly for increasingly complex, safety-critical payload. Attempts have been made to increase flexibility in these protocols, e.g. FlexRay and TTCAN, however, this, so far, has only been possible for information that is not safety-critical.

In this chapter, INCUS was designed to improve the channel utilisation by allowing the slot length of nodes to be configured in accordance with the actual payload requirements of these nodes inside a TDMA round. While my proposed approach is based on the traditional TDMA scheme utilised in the Time Triggered Architecture, it significantly improves bandwidth utilisation over the traditional schemes. The analyses performed in this chapter have shown that INCUS reduces the gross overhead time by almost 90%, improving overall bandwidth utilisation efficiency almost twofold in a typical automotive brake-by-wire system scenario.

INCUS+ with further enhancements in the INCUS, significantly improves channel utilisation over FlexRay, while guaranteeing atomicity and safety at the protocol level. INCUS+ achieves increased channel utilisation by allowing the slot length of each node to be configured in accordance with its actual transmission payload requirements for each TDMA round of a cluster cycle. This eliminates node slot idle times for all nodes, hence reduces transmission overhead. Compared to FlexRay which is a TTA-based communication protocol for safety-critical real-time systems, this significantly improves bandwidth utilisation. In the analysis, we have seen that this kind of flexibility makes it possible to reduce the gross overhead time by almost 99%, improving overall bandwidth utilisation efficiency almost nine times compared to FlexRay in an autonomous vehicle system case study. An implementation of multiple operational modes each with different communication schedule further enhances the flexibility of INCUS+. Despite the added flexibility, the same level of predictability (predefined schedules for channel access) has been maintained. This is crucial to ensure the safety of the system at the communication protocol level, and for implementing critical services such as the membership service and clock synchronisation. My approach not only increases flexibility and channel utilisation for safety-critical payload, but also maintains the ability to handle faults in a fail-silent way, at the same level as FlexRay and other TTA-based protocols.

Chapter 5

Software Architecture Design and Implementation

Engineering real-time communication protocols is a complex task, particularly in the safety-critical domain. Current protocols exhibit a strong tradeoff between flexibility and the ability to detect and handle faults in a deterministic way. Model-driven engineering promises a high level design of verifiable and directly runnable implementations. Arrangements of Logic-Labelled Finite-State Machines (LLFSMs) allow the implementation of complex system behaviours at a high level through a subsumption architecture with clear execution semantics. In this chapter, it is shown that the ability of LLFSMs to handle elaborate hierarchical module interactions can be utilised towards the implementation of testable, safety-critical real-time communication protocols. This chpater presents an efficient implementation and evaluation of INCUS+, a time-triggered protocol for safety-critical real-time communication that transcends the rigidity imposed by existing real-time communication systems through the use of a high-level subsumption architecture.

5.1 Engineering a Software Architecture for Safety-Critical Real-Time Systems

Nowadays, a model-driven software development approach is widely used by developers in contrast to lower level implementation approaches, as it assists in developing, faster and simpler modules and applications [149]. Finite State Machines (FSMs) or Behaviour Trees are used to represent high level specifications of behaviours. This kind of modelling approach fulfills the agenda of Model Driven Engineering [150–153] for software development. In contrast to other, more traditional implementation approaches, Logic-Labelled Finite-State Machines (LLFSMs) [154] allow translating requirements into high-level, executable models [155–157]. These are less susceptible to implementation errors as models can be directly interpreted, simulated, verified, and executed on a large number of platforms, including embedded control systems [158–160].

In control systems, where different modules are interacting with each other, it becomes very important to predict the results and shield the details of one module from others. To solve this problem, the *subsumption architecture* [161] has proposed behaviour-based decomposition of such complex systems into layers of increasing level of abstraction, where high level layers can subsume the lower level layers. Several other similar approaches [162–164] were developed, but one big advantage of the subsumption architecture is the ability to cater for the evolution of the complexity of a control system by accretion of higher-level layers. This approach allows the incremental development of a control system, as the addition of each new layer provides a new additional behaviour to the controller. Further layers can be added on top of the existing layers without affecting their behaviours. This way, a functional controller will always be available with each new behaviour throughout the development process. So far, the subsumption architecture has largely been used to build complex embedded systems, such as robotic control systems [165–171].

In this chapter, LLFSMs are used as a modelling tool, where transitions from one state to another state are based on expressions in logic rather than events. This not only reduces the overhead significantly as, for example, no memory allocations are required for event queues, but also makes system performance fully predictable. Although modelling with LLFSMs is a very effective approach as shown in the literature [172, 173], to my knowledge, no attempts have been made to date to use them towards the implementation of a safety-critical, hard real-time system. This chapter not only discuss the implementation of proposed protocol using LLFSMs, but also shows how the subsumption architecture helps prevent design issues and how an arrangement of LLFSMs has proven a better technique that enables to design and develop a more complex protocol faster.

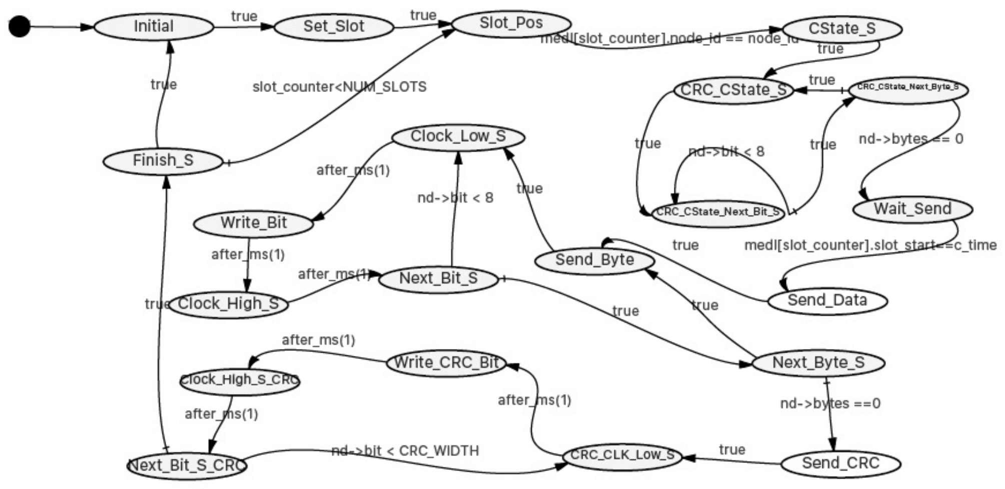

FIGURE 5.1: Transmission behaviour of INCUS+ using an individual LLFSM. This machine contains around 50 states (not all are shown here) without the implementation of other protocol services such as behaviour for the node start-up and reintegration, clock synchronisation, the membership service, mode changes, and other FTAMs.

5.2 Executable Communication Model

When implementing a communication protocol, a key design decision is the choice of tools and the level of modelling for this implementation. The prototype software for TTP/C, for example, was designed at a low, procedural level and implemented using a mix of C++ and assembly language [15]. While this approach certainly offers predictable, high performance (short only, perhaps, of a direct hardware implementation), a key disadvantage is the design and development effort (several man years) required by such an approach. Moreover, low-level software is often wedded to a specific hardware and difficult to port to a different platform. Nevertheless, to date, the rigorous timing requirements that need to be modelled early on in the design process has made it difficult to model verifiable executable real-time behaviour at a high level [16]. These and other difficulties often encountered with high level engineering of software has often prompted the question of whether it makes sense to engineer software, and hence, there is now a trend to view engineering as *craft supported by theory*, leading to best practices in software engineering [174]. A common element in software architectures in general, but particularly in control software, are finite-state machines. In fact, the most commonly used artifact for the description of software behaviour in UML are state diagrams [175, 176]. Logic-labelled finite-state machines (LLFSMs) are turing complete, making them fundamentally equivalent to any mechanism to model system behaviour, with key advantages [16] that shall make them the preferred model here. First, they offer a very clear semantics of concurrency, tremendously simplifying the cognitive burden for the developer [16]. Importantly, though, their execution semantics much more closely resembles the principles of the TTA [177] and is in direct contrast with the optimistic best-effort approach of event-driven systems [16]. INCUS+ is implemented through executable models using LLFSMs, and so far, this is the first attempt at implementing software suitable for safety-critical hard real-time systems using this approach.

5.2.1 LLFSM Design of INCUS+

In the first iteration, all the lower level and higher level implementation details are embedded in a single LLFSM. To this end, INCUS+ is modelled in stages, starting from the very basic functionality of the protocol, i.e., transmit and receive message at a pre-defined point of time, following the time-triggered approach [178] of the specification. Then, basic fault tolerance algorithms and mechanisms (FTAMs) were added, e.g. the Cyclic Redundancy Check (CRC) whether a received message is corrupted. Step by step, incrementally added additional behaviours required by the protocol to take it towards the full specification. Each of the steps was designed using `MiCASE` [179] and compiled to an executable that was run, tested, and verified using `clfsm` [179]. What is important to note is that, since LLFSMs represent executable models, each of these steps, despite not yet implementing the full specification, gave a fully functioning, executable prototype that made it possible to simulate, exectute, test, and validate.

The complete INCUS+ specification is described in Chapter 4, but here, the focus will be on one key aspect and briefly discuss the implementation of the transmission behaviour of INCUS+ using a single LLFSM as shown in Figure 5.1. In the `Initial` state, all the necessary parameters are initialised such as the Message Descriptor List (MEDL) that holds the time schedule for the data transmission and reception phase for all nodes. Each node has an identical copy of the MEDL. All nodes starts from slot *zero* as their first slot in the `Set_Slot` state, then transitioning straight to the `Slot_Pos` state. In this slot, each node will check the MEDL to figure out whether it needs to act as a sender node or receiver node according to the current slot position. Note that, this section discusses the message transmission mechanism of INCUS+, and ignores the receiver part (as a receiver node) in this example, but the receiver follows analog steps to the transmitter states discussed below. So if, in the current slot, the node is meant to act as a sender node, a corresponding transition is made from the `Slot_Pos` state to the `CState_S` state. The Controller State (C-State) is initialised in this state so that CRC value can be calculated over the C-State. This allows a message to be rejected as incorrect, not only if there is a physical transmission error, but also if there is any other fault that causes C-State disagreement [15].

After initialisation of the C-State, the node transitions to the next state `CRC_CState_S`. From there, it takes a byte at a time and calculates the CRC on each individual bit of that byte and when finished, takes the next byte. This continues until there are no more data left for transmission. This whole procedure is achieved through the state transitions from the `CRC_CState_S`, `CRC_CState_Next_Bit_S`, and

CRC_CState_Next_Byte_S states. The next state after the CRC calculation is the Wait_Send state, from which a transition is made to the Send_Data state, only once the time at sender node is equal to the time defined in the MEDL for the actual message transmission. This time is termed the *slot start time* and implements the essential trigger time for the sender node.

It is important to note that, other than the conditional transitions shown in the figure, there is no conditional code here, making, together with the deterministic scheduling of clfsm [16], the execution time of the compiled code extremely predictable in correspondence with the structure of the high-level model. In fact, as a consequence, the worst-case execution time (WCET) is guaranteed to be the same as the best-case execution time, minimising the temporal jitter of the transmission start state. In this state, the Send_Data state, the original message and number of bytes of the message are fetched from the MEDL and then the next state is the Send_Byte state. This state takes one byte of the message at a time, and transmits it bit by bit (through the Clock_Low_S, Write_Bit, and Clock_High_S states, also updating the CRC at each step), and then looks for the next byte (Next_Byte_S).

Once no bytes are left to transmit (`nd->bytes == 0`), the sender transitions to the Send_CRC state, where all the bytes of the CRC are transmitted. The mechanism of CRC transmission is same as the transmission of the original message and the states used for transmitting CRC value (CRC_CLK_Low_S, Write_CRC_Bit, Clock_High_S_CRC, and Next_Bit_S_CRC), have analogous functionality to the data transmission states above, but transmit the CRC instead. The Finish_S state concludes the cycle, incrementing the MEDL slot position and transitioning straight back to Slot_Pos if there are more MEDLs slots to operate on, or back to Initial, if the end of the TDMA cycle has been reached and the above steps repeat from the beginning of the MEDL (slot zero).

One pattern that becomes apparent in this initial design is a replication of concerns. In other words, despite the fact that the above description only details the transmission phase of the protocol, this already has replicated the relevant states used in message transmission, i.e., the states required for transmitting the CRC essentially mirror the states used for transmitting the message payload. The states representing the receiver very much mirror the sequence described above, with only minor differences, such as the provision of a small receive window to compensate for clock drift and jitter. Up to this point, the complete LLFSM, including the receiver, already contains *fifty states* and it has not yet implemented important parts of the communication protocol specification, such as the behaviour for the node start-up and reintegration, clock synchronisation, the membership service, mode changes, and other FTAMs such as the detection of transmit and receive errors on the basis of different timeout parameters. As this single LLFSM grows bigger, it becomes more complex and was nearly impossible to add remaining behaviours of the protocol by adding and replicating more states. We will discuss how this has been addressed through subsumption in the next section.

Nevertheless, this initial implementation already serves as a very important proof that it is not only possible to implement a protocol for safety-critical real-time systems using LLFSMs, but also that this design and implementation process can be done much more rapidly (several weeks vs. a few man-years) at a high level, yet yielding fully executable models at every stage. This leads to the next stage of considering a refined approach that greatly enhances the modularity of the design.

5.3 INCUS+ Subsumption

To reduce the complexity of the overall design and increase the modularity, my implementation follows the principles of the *subsumption architecture* [161], which allows to split out functionality into modules that can hierarchically be subsumed by higher level modules. Arrangements of LLFSMs allow the implementation of a subsumption architecture by integrating a number of different finite-state machines, each forming a component or module that can be deactivated using a `suspend` operation or activated using a `resume` or `restart` operation[1] [173].

[1]With LLFSMs, the `restart` operation simply restarts the machine from its `Initial` state, while the `resume` operation resumes from the previously active state.

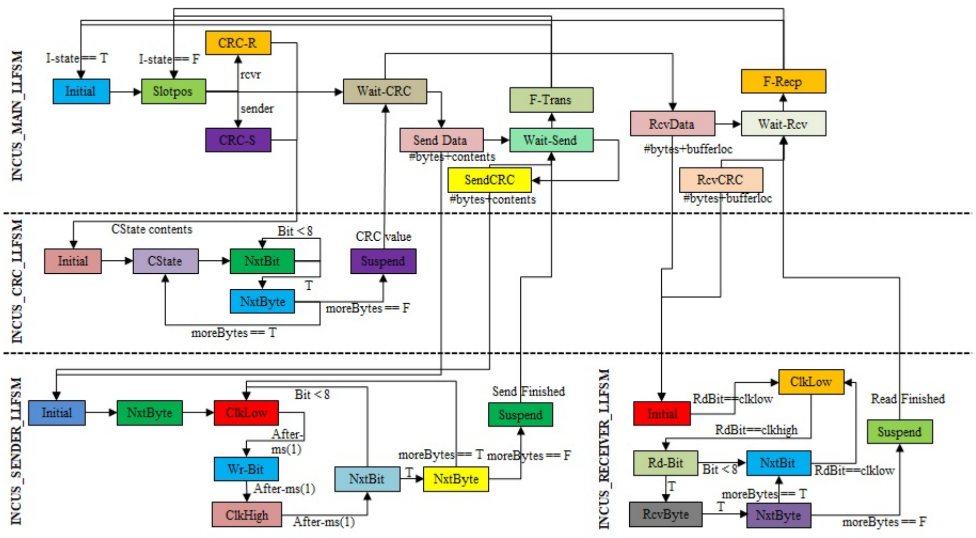

FIGURE 5.2: This figure shows the subsumption architecture of INCUS+ where single machine is splitted into number of multiple LLFSMs

State machine vectors formed by an arrangement of LLFSMs make the decomposition of sub-behaviours into modules particularly straightforward. This process already identifies repetitive the sub-behaviours, such as the transmission of CRC vs. payload data. Splitting these elements out into individual modules is as simple as factoring out those states into an individual sub-machine. A decomposition of the earlier, single LLFSM implementation of INCUS+ into the following four sub-LLFSMs is shown in Figure 5.2:

1. INCUS+_MAIN_LLFSM
2. INCUS+_CRC_LLFSM
3. INCUS+_SENDER_LLFSM
4. INCUS+_RECEIVER_LLFSM

The INCUS+-MAIN-LLFSM module acts as a high-level master-LLFSM that actually controls the behaviour of the other sub-LLFSMs. These sub-machines are composed of the CRC-LLFSM, SENDER-LLFSM, and RECEIVER-LLFSM machines. The Main LLFSM runs concurrently with the sub-LLFSMs and only has the principle purpose of implementing the high-level stages of the protocol and to suspend and restart the sub-machines. These sub-machines are the modules

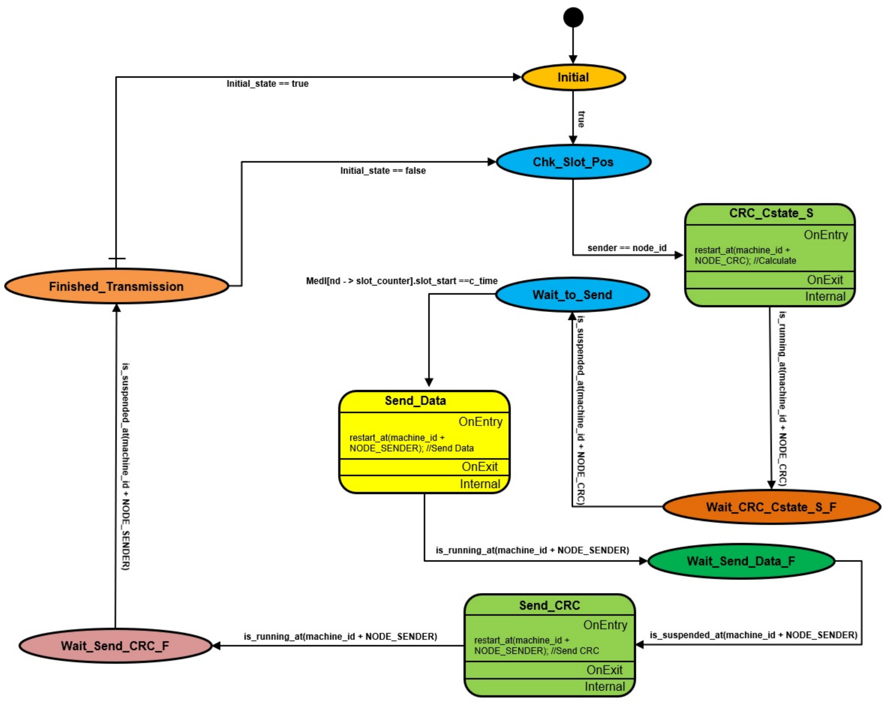
Initial
true
Chk_Slot_Pos
Initial_state == true
Initial_state == false
CRC_Cstate_S
OnEntry
restart_at(machine_id + NODE_CRC); //Calculate
OnExit
Internal
sender == node_id
Finished_Transmission
Medl[nd - > slot_counter].slot_start ==c_time
Wait_to_Send
Send_Data
OnEntry
restart_at(machine_id + NODE_SENDER); //Send Data
OnExit
Internal
is_suspended_at(machine_id + NODE_CRC)
is_running_at(machine_id + NODE_CRC)
Wait_CRC_Cstate_S_F
is_suspended_at(machine_id + NODE_SENDER)
is_running_at(machine_id + NODE_SENDER)
Wait_Send_Data_F
Send_CRC
OnEntry
restart_at(machine_id + NODE_SENDER); //Send CRC
OnExit
Internal
is_running_at(machine_id + NODE_SENDER)
is_suspended_at(machine_id + NODE_SENDER)
Wait_Send_CRC_F

FIGURE 5.3: INCUS+_MAIN LLFSM: This figure shows Main LLFSM of INCUS+. This only invokes Sender and CRC machines.

in the subsumption architecture that implement the corresponding underlying behaviours, when required.

In the previous section(5.2.1), the example of message transmission in INCUS+ using a single LLFSM was used to highlight the issue of design complexity. In the following section, I will demonstrate how this complexity issue is tackled by implementing the message transmission behaviour of INCUS+ using the subsumption architecture.

5.3.1 Tackling Design Complexity using Subsumption

The implementation of the transmission behaviour of INCUS+ using the subsumption architecture is done by decomposing the single LLFSM from Figure 5.1 into the three sub-LLFSMs 1–4, i.e., INCUS+_MAIN-LLFSM, INCUS+_CRC-LLFSM, INCUS+_SENDER-LLFSM, and INCUS+_RECEIVER-LLFSM. Figure 5.3 shows the main machine. The main machine acts as a master LLFSM and can run concurrently with the sub-LLFSMs. While the subsumption architecture allows multiple

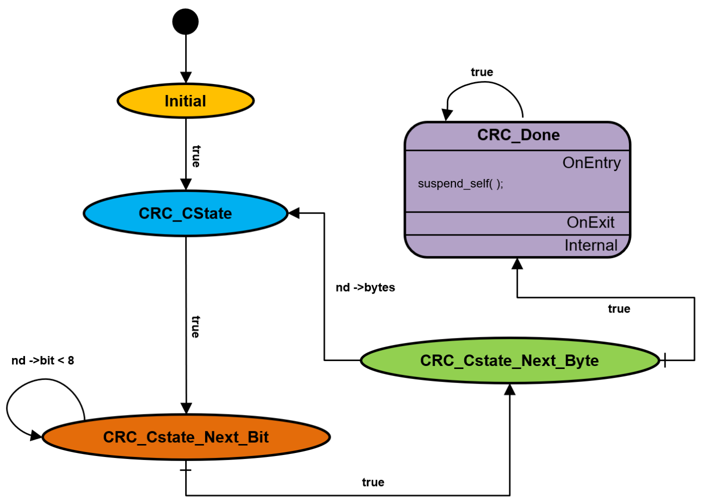

FIGURE 5.4: INCUS+_CRC LLFSM: This Machine is used to calculate the CRC value before transmitting and after receiving a frame to and from a communication channel. This machine is invoked by Main LLFSM machine at different instants.

sub-machines to operate concurrently, and while the execution semantics of LLFSMs is clearly defined to avoid concurrency issues or temporal inconsistencies [16], this approach deliberately kept the design of the INCUS+ implementation simple, not requiring the concurrent operation of multiple sub-machines at the same time. This greatly simplifies WCET measurement and further reduces the design and validation complexity of the system.

The overall transmission mechanism follows the steps discussed in Section 5.2.1. After initialising the relevant protocol parameters in its `Initial` state and verifying, in the `Chk_Slot_Pos` state, whether the current node is the sender node. If the node is the current transmitter, it transitions to the new state `CRC_CState_S`. To perform the CRC calculations over the local C-State prefixing the payload transmitted in the message, the main LLFSM will now activate the CRC module (Figure 5.4) by using `restart_at(machine_id+NODE_CRC)`.

The CRC LLFSM runs concurrently with the main machine and performs the CRC calculation over the data referenced by the main LLFSM. In the case of the `CRC_CState_S` state of the main machine, the CRC data reference points to the C-State that the CRC shall be calculated over. The states of the CRC LLFSM are same the ones described in Section 5.2.1, but the clear advantage here is, that there is no need to replicate these states multiple times, whenever it is required to perform a CRC calculation. In fact, this calculation is irrespective of the node's current role as a sender or receiver, and thus, unlike in the previous implementation, no further replication is necessary.

While the main machine technically runs concurrently with the CRC module, activating the CRC module does not require any concurrent operation, so the main LLFSM simply transitions to the `Wait_CRC_CState_S_F` state (through the transition labelled `is_running_at (machine_id+NODE_CRC)`) where it waits for submachine completion through use of the `is_suspended_at(machine_id+NODE_CRC)` predicate. To notify completion through this predicate, the CRC LLFSM will simply suspend itself by using `suspend_self()` in its `CRC_Done` state after having completed calculating the CRC value. This is semantically equivalent to subsumption akin to the UML sub-machine notation [173].

As soon as the CRC calculation has concluded, the main machine transitions to the `Wait_to_Send` state, where it waits for the arrival of the transmission slot action time. To transmit the message, the main LLFSM now activates the sender LLFSM (Figure 5.5), while again simply waiting for completion by sitting idle in the `Wait_Send_Data_F` state until the sender LLFSM has completed and suspended itself. Unlike the example from Section 5.2.1, where the transmission logic had to be replicated for transmitting the message CRC, the same sender LLFSM can now be used and will be restarted by the main machine when in order to send the CRC value as implemented by the `Send_CRC` and `Wait_Send_CRC_F` states in the main LLFSM. So contrary to the implementation of transmission behaviour using the single LLFSM, the subsumption architecture eliminates the complexity imposed by state-replication, while maintaining the ability to implement real-time behaviour following the same, conceptual design principles.

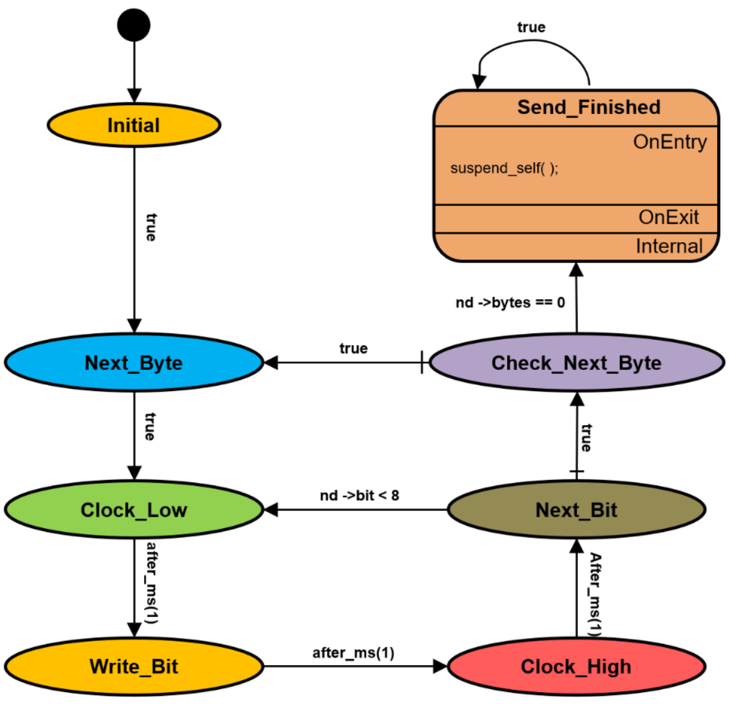

FIGURE 5.5: INCUS+_Sender LLFSM: This figure shows the implementation of a Sender machine. This machine is used to transmit data frame and its CRC value. This machine is invoked by Main LLFSM machine at different instants.

5.3.2 Adding new Behaviours using the Subsumption Architecture

This section will now briefly describe the subsequent, iterative steps that were conducted using the subsumption approach. It would have been hard to continue the modelling of INCUS+ using a single LLFSM due to the complexity explosion alluded to earlier. In the following analysis I will show how straightforward it now is to add a new behaviour while retaining all the functionality of the existing behaviours of INCUS+ using the subsumption architecture and arrangements of LLFSMs.

5.3.2.1 Start-up and Re-integration of nodes

So far, it was assumed in the implementation that all nodes successfully resolved their start-up collision scenario [8] and they are in the state where they have synchronised clocks and are ready to transmit/receive messages. To implement

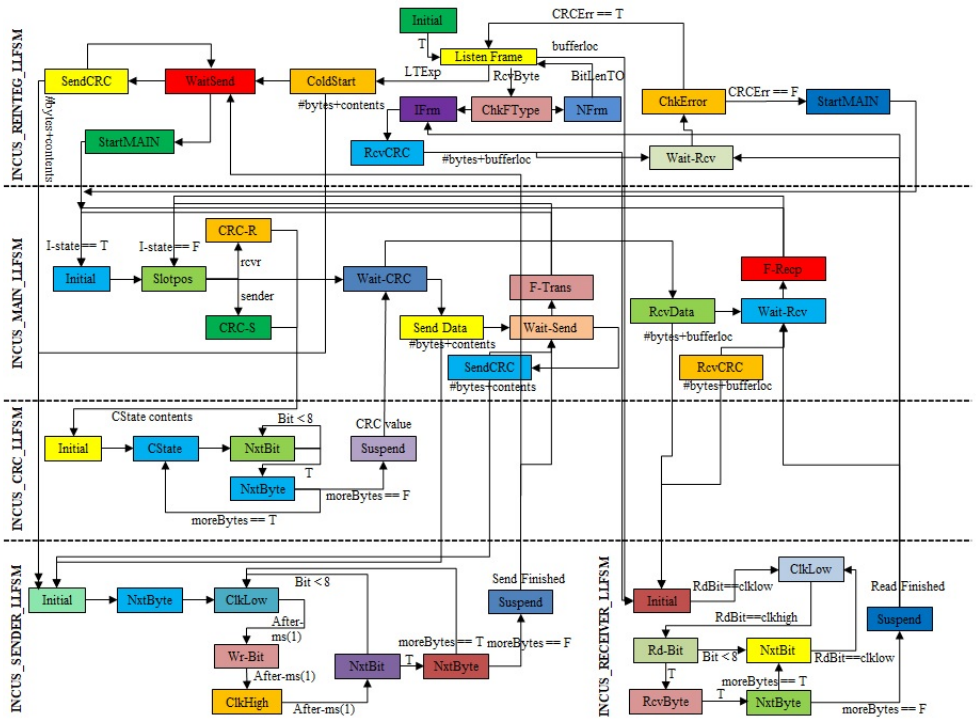

FIGURE 5.6: This figure shows the addition of new behaviour using subsumption architecture in INCUS+.

system startup and reintegration in accordance with the INCUS+ specification, this need to add another sub-LLFSM as shown in Figure 5.6. This machine is named RE_INTEGRATION LLFSM in a sense that this LLFSM is, in the fault-free case, used only once to run the start-up scenario when all the nodes are turned-on initially. After this, the main LLFSM is used most of the time, but has the ability to trigger a restart to re-integrate a lost node to the cluster of nodes in case of a fault that requires re-integration. The re-integration LLFSM acts as a master-LLFSM only when all nodes are turned-on the first time. It will suspend all other sub-LLFSMs and runs the node start-up algorithm. Once all the nodes are up, the RE_INTEGRATION-LLFSM will suspend itself after starting the main LLFSM. Now the sphere of control shifts to the main machine as above, which will perform the normal operation of the protocol as described. Importantly, a comparison between Figure 5.2 and Figure 5.6 shows that addition of this new behaviour has been achieved by just adding a single new layer on top of INCUS+_MAIN_LLFSM. Most importantly, the main machine did not require any modification or change to the structure of the previously existing layers of

LLFSMs.

5.4 Summary

This chapter presented my design and implementation of INCUS+. Initially, I designed the protocol without using a subsumption architecture. One pattern that became apparent in this initial design is a replication of concerns. A single LLFSM is used to implement the transmitter and receiver functionality. This replicated the relevant states used in message transmission, i.e., the states required for transmitting the CRC essentially mirror the states used for transmitting the message payload. The states representing the receiver very much mirror the sequence described above, with only minor differences, such as the provision of a small receive window to compensate for clock drift and jitter. As this single LLFSM grows bigger, it becomes more complex, making it nearly impossible to implement more complex behaviours of the protocol by adding and replicating more states. This also made debugging and validation nearly impossible. Due to this complexity, some simple errors, for example, statements written mistakenly at *OnEntry* of a state instead at *OnExit* and vice versa, took weeks to be detected. This lead to the consideration of a refined approach that greatly enhances the modularity and decomposability of the design. In summary, my implementation has shown that a high-level implementation of a communication protocol for safety-critical real-time systems based on the subsumption architecture is not only possible, but facilitates the incremental development of the system using executable models throughout. This shows that with INCUS+ implementation, based on an arrangement of multiple LLFSMs, the scope of the subsumption architecture is not limited to modelling the behaviours of traditional control systems, but this can also be used to develop finite-state machines with predictable execution semantics and timing. This chapter demonstrated that LLFSMs support system development of INCUS+ in an iterative way or in stages, where it can execute, test and refine a safety-critical real-time system at a given level before starting a new level. This made it possible to ultimately implement a more flexible communication protocol in comparison with existing TTA based communication protocols, where the implementation, refinement, and validation was a lot more complex. The modelling technique presented in this chapter has been shown to make feasible designing flexibility into communication protocols with the strict predictability and timing required by dependable real-time systems. Furthermore, this has shown that the complexity of state-replication can be avoided very effectively by using the subsumption architecture provided by arrangements of LLFSMs when developing a communication system, without losing the fundamental properties of predictable real-time performance. The subsumption architecture made it possible to incrementally refine the implementation by adding, modifying, or changing the behaviour of a sub-system without interfering with unaffected components of the system.

Chapter 6

Ensuring Fail-Silence

As INCUS+ is built on the broadcast semantics of the Time-Triggered Architecture [6], nodes need to cooperate to ensure collision-free communication. Therefore, it is imperative for any Safety-Critical distributed Real-Time (SCRT) system to prevent the communication channel from being monopolised by a faulty node [180][8]. My protocol is based on a time-triggered communication schedule, where each node broadcasts its messages according to a pre-determined transmission pattern. Therefore, a faulty node can corrupt the communication over the communication channel by transmitting its messages out of schedule, at arbitrary points in time [180]. This can lead to a babbling idiot failure [180]. To prevent possibility of such a failure, I propose a bus guardian approach that introduces an independent arbiter [180] for bus access. In a bus topology, the bus guardian is a device attached with each node in the SCRT system, whereas in a star topology, a redundant bus guardian pair can be installed to shield the whole system from babbling idiot faults [8]. In this chapter, a bus guardian approach is presented to enforce a fail-silent behaviour of the participating nodes in the time domain. A formal verification model is used to verify the transmission window timings of a transmitter node along with its bus guardian and receiver windows. In particular, I will formally verify that the design rules for the timing parameters of transmission windows used in INCUS+ enforce the required temporal properties. This further verifies that a message transmitted by a non-faulty transmitter can never be blocked by a non-faulty bus guardian and will be accepted by a non-faulty receiver. Any transmission slot overlap is also ruled out through this model.

6.1 Introduction

In a distributed real-time system using a broadcast mechanism for communication, a node that is transmitting at arbitrary points in time can pose a serious threat of communication failure [181]. Nodes that exhibit such behaviour are known as babbling idiots [180][182]. These faulty nodes are not obeying the medium access rules imposed by the underlying communication network and consequently corrupt the transmitted frames by non-faulty nodes [180][182]. The operation of an SCRT system under the presence of these babbling idiot faults can lead to lack of dependability and a high cost of point-to-point network topologies [182]. An ideal case would be if a node exhibited only a single failure mode. A node is considered fail-silent only if it either produces a correct result or no result at all [183]. A node can be regarded as a separate fault containment region through its fail-silent behaviour, which means that a fault produces by a node in the system cannot propagate to other nodes in the system [184]. In a time-triggered environment, channel access is based on the progression of a global timebase, where clocks of all participating nodes are synchronised with each other [177]. A transmission slot is allocated to each node and it has an exclusive access to the communication channel during this allocated time window [6]. The channel access pattern of each node in a time-triggered communication is defined in the time domain and thus, a node that violates this access pattern is termed a babbling idiot, as it may talk over the transmission of a node that is legitimately scheduled to transmit at the point in time of the faulty node's interference [8, 180].

Time-triggered communication uses a communication schedule with slots held by each node in advance [6]. This schedule records the identity of the transmitter node for each slot as well as the slot length and transmission start time [8]. The transmitter node starts its transmission some time after the slot start time and finishes its transmission some time before the slot duration has elapsed [180][8]. The receiver nodes start listening for incoming traffic at the beginning of the slot and ends when the slot duration has expired [180][8]. The timings of the communication activities both at transmitters and receivers are driven by their respective local clocks[8][68]. However, these clocks are synchronised among the participating nodes within some precision threshold [180]. It is impossible for these nodes to exhibit the exact same clock value at every given instant [180]. Therefore, a clock skew may cause a transmitter to transmit before some receivers start listening to the network or finish its transmission when some receivers already have stopped listening to the traffic on the network [180]. This could lead to a failure where some nodes accept the transmission while other rejects it [180][8]. To avoid such failures, slot timing parameters for transmitter and receiver nodes must be chosen with care to prohibit such failure scenarios[8]. Typically, this is achieved by introducing a small window of tolerance that accounts for the jitter introduced by the fundamental limits of precision in clock synchronisation[8]. In addition to this, bus guardian windows must be set so they will block the faulty transmitter from transmitting outside its allocated slot timings[8]. We will formally verify the slot timing parameters for transmitters, receivers, and bus guardians in this chapter.

6.2 Bus Guardian

The Bus Guardian (BG) is an autonomous subsystem that protects the communication channels from temporal transmission failures [8]. Additionally, the BG can serve to protect the bus against transmissions that are apt to lead to ambiguous results at the receivers (so-called ‘slightly-off-specification’ faults) [8]. The BG functionality may be implemented locally or centrally, but importantly the BG needs to be independent from the node(s) it protects[8]. In any case, a node must have a BG to achieve fail-silence in the temporal domain [8]. Such a local BG only has the knowledge of its node‘s schedule (sending slot) and must use an independent external clock source [8, 185].

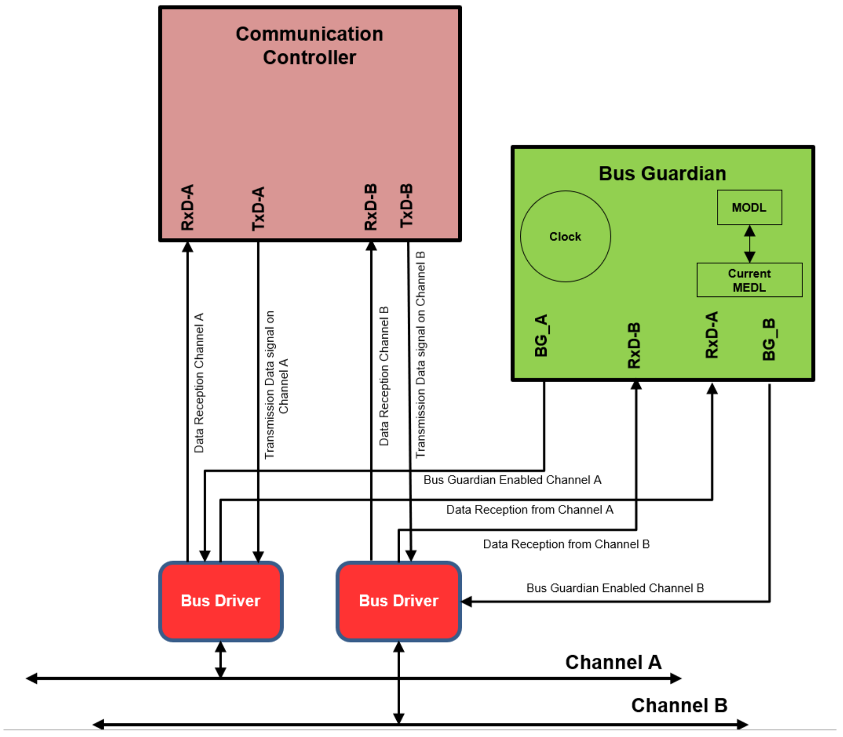
Communication Controller
RxD-A
TxD-A
RxD-B
TxD-B
Bus Guardian
Clock
MODL
Current MEDL
BG_A
RxD-B
RxD-A
BG_B
Data Reception Channel A
Transmission Data signal on Channel A
Data Reception Channel B
Transmission Data signal on Channel B
Bus Guardian Enabled Channel A
Data Reception from Channel A
Data Reception from Channel B
Bus Guardian Enabled Channel B
Bus Driver
Bus Driver
Channel A
Channel B

FIGURE 6.1: Independent Bus Guardian and Communication Controller

The issue of babbling idiot faults is handled with independent bus arbitration, where each node is equipped with an independent BG in a bus topology [8].[1]

6.2.1 Bus Guardian Architecture

The design of a BG requires that no common mode of failure can occur between the guardian and the node it is protecting [185]. A common mode of failure means both the node and its BG fail at the same time and thus, there would be no protection provided by such BGs [185][8]. Therefore, implementing a BG as a separate device with no hardware dependency can avoid common mode of failures [185]. There are number of BG architectures in the literature specified as follows:

[1]Please note that while a bus topology is used here, at a protocol level it only require broadcast semantics. Therefore, the same applies to other topologies with equivalent semantics, such as a star topology.

6.2.1.1 Closely coupled bus guardian

In this architecture, the BG is used to prevent the node from transmitting too frequently on the communication channel [185]. The BG is using additional logic to prevent a faulty node to transmit outside its schedule window [185]. However, this type of guardian provides the actual interface to the network and therefore, a fault in BG itself cannot protect from babbling idiot failures [185]. This issue is tackled through redundancy but independence cannot be guaranteed as both are closely coupled[186]. This architecture is used in Delta-4 [183] and CAN [187].

6.2.1.2 Loosely coupled bus guardian

In this architecture, a node is listening to the channel but message transmission is only allowed if the associated BG permits the node to transmit over the communication channel [185]. Bus drivers are disabled and a node has to inform the BG before attempting to transmit on the bus [185]. If the BG agrees with the node's request, then it will enable the bus driver [185]. This BG architecture is used in TTP [181] and FTMP [188]. However, these guardians are not using independent clocks, which can lead to a common mode of failure.

6.2.1.3 Independent bus guardian

The previously existing communication protocols can use a simple design of bus guardians, and even dumb guardians that only listen to transmission patterns at the beginning and then follow the same patterns, as the communication schedule does not change during the entire operation. Here, I propose an approach that is using an independent BG design as shown in Figure 6.1. This design, importantly, has its own clock and a copy of the complete transmission schedule for all TDMA rounds as well as different operational modes defined in Mode Descriptor List (MODL) for its respective node, and listens for the incoming traffic at specific instants (defined in the Message Descriptor List (MEDL)). The guardian prevents the bus from monopolisation by a sending node which is trying to transmit more often than it is scheduled for during a TDMA round. This also prevents the Communication Controller (CC) from transmitting outside its allocated transmission window. The bus guardian is a completely separate device that is directly connected to the bus. While this leads to a more complex design, crucially, it supports the flexibility of dynamic communication schedules. Lets have a look at the different types of incoming traffic and evaluate how the independent bus guardian can perform when listening to it.

1. **Listening to a transmission pattern in the first few TDMA rounds:** These types of bus guardians are known as dumb guardians. They just listen to the traffic in the first few TDMA rounds and configure their window accordingly. This approach can work well if an SCRT system only supports static and equal length time slots for all nodes. This approach can support INCUS only in a bus topology where a guardian node can learn the transmission pattern of its associated node and the same pattern repeats in other TDMA rounds. Please note INCUS supports variable slot lengths inside a TDMA round, but this pattern can be learnt.

2. **Listening to transmission patterns defined in a single MEDL:** These bus guardians have a copy of the MEDL and have complete knowledge of the communication schedule. These bus guardians can work with INCUS as well as its enhanced version INCUS+ with single mode of operation, but with different schedules and in different TDMA rounds. This approach can work well if nodes are successfully integrated into the cluster and follow the predetermined transmission patterns. To support full functionality of INCUS+ bus guardians need to listen different types of frames as discussed in the previous chapter.

3. **Listening to Initialisation Frames (I-frame):** Bus guardians have knowledge of the MEDL, but in case a node is lost it gets reintegrated into the cluster by successfully listening to an I-frame. This is necessary as a re-integrating node must know about the current TDMA round and slot position. Similarly, if a bus guardian is lost and re-starts itself in a

healthy state then it must reinitialise itself with the current slot position and TDMA round. This is only possible if the bus guardian is listening to I-frames. However, this mechanism does not support the full functionality of INCUS+, when using different operational modes with different transmission patterns. Please note, the fail-silence behaviour is ensured by a redundant bus guardian if a bus guardian fails. Please see the details in Section 6.2.2.

4. **Listening to full traffic:** The bus guardian is configured to listen to all the incoming traffic to support the full functionality of INCUS+. As in INCUS+ different modes of operations are supported and each mode can support different communication schedule, this means just listening to I-frames is not enough. A mode change request can be launched at any time. This can be an immediate or a deferred mode change, as discussed in the previous chapter. This mode change request will be embedded in the header of a normal frame. The Controller State (CState) [8] of a requesting node will inform the other node during a normal operation about the mode change request. To simplify the design, only specific nodes in the cluster are allowed to make such a request. Before agreeing on the requested mode, all the receiver nodes perform a check to determine if the requesting node is allowed to make such a request. If the result is negative then the request is ignored and the requesting node is removed from membership. Bus guardians cannot protect the channel if they are unable to listen the full incoming traffic if the timing of transmission depends on the operational mode. This design is more costly if a bus topology is used, where each node needs to have a completely independent guardian and a redundant guardian for the second bus as well to avoid a single point of failure. However, this can be implemented in a more cost-effective way by using a star topology with a central and independent bus guardian along with a redundant guardian to avoid single point of failure.

6.2.2 Fail-Silence through redundancy in different network topologies

In the previous section, we saw that there is a possibility that a bus guardian itself gets faulty and thus, becomes unable to protect the communication channel from babbling idiots. To avoid such conditions, different topologies can be used with reduced to full redundancy levels [8]. Figure 6.2 shows the architecture of maximum redundancy in a bus topology,

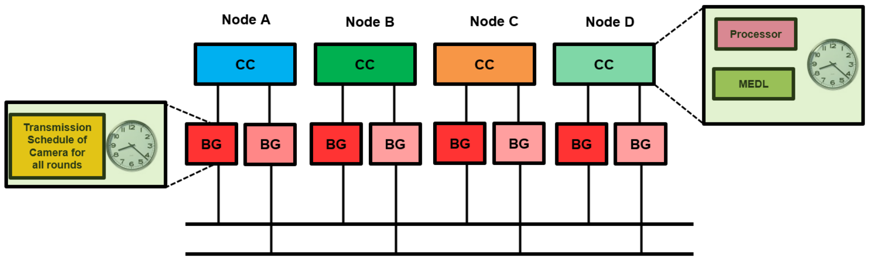

FIGURE 6.2: Bus Guardian layout in bus topology with full redundancy

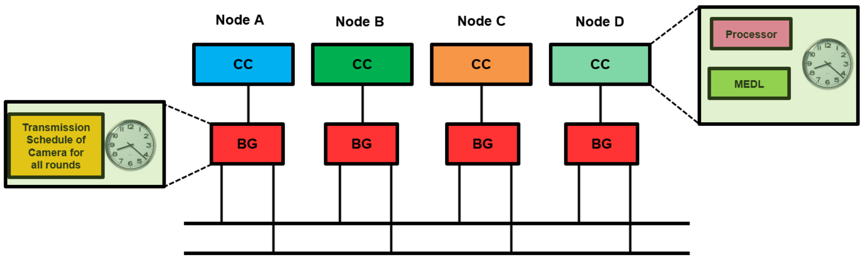

FIGURE 6.3: Bus Guardian layout in bus topology with reduced redundancy

where each bus is protected from each node by a separate BG. In case of a BG failure, a node is still able to transmit on the other channel. However, since critical nodes have to be replicated to avoid a single point of failure, a simpler design can then be used, where only a single BG is used for each node over the redundant communication channels as shown in Figure 6.3.

In a star topology, by comparison, a single BG for all nodes as shown in Figure 6.4, would introduce a single point of failure and in-case of a fault in the guardian, traffic on both channels would be interrupted. Therefore, the required architecture would be star couplers in a redundant star topology as shown in Figure 6.5.

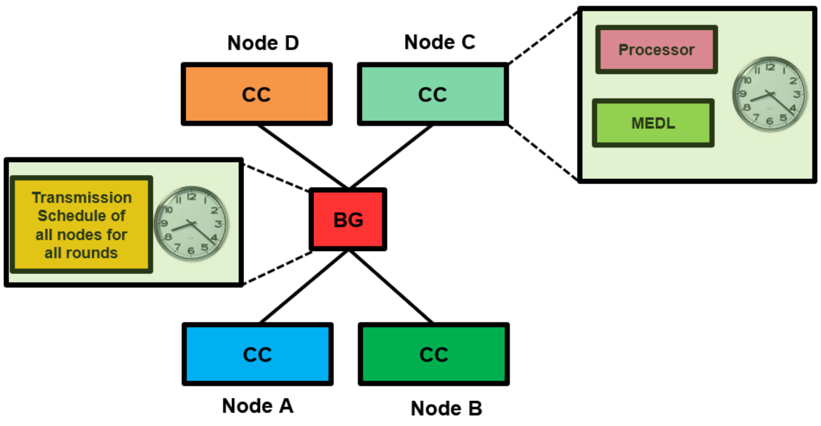

FIGURE 6.4: Bus Guardian layout in star topology with reduced redundancy

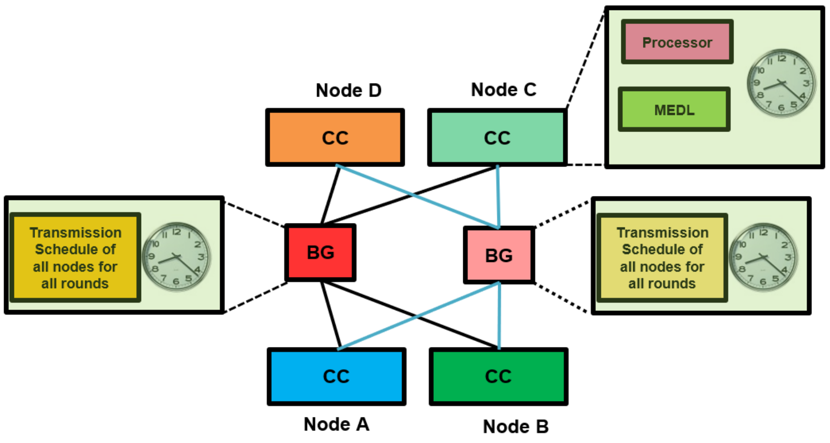
Processor
MEDL
Node D
Node C
CC
CC
Transmission Schedule of all nodes for all rounds
BG
BG
Transmission Schedule of all nodes for all rounds
CC
CC
Node A
Node B

FIGURE 6.5: Bus Guardian layout in star topology with full redundancy

A bus guardian in the bus topology, that only uses a periodic signal from its respective node to synchronise its clock with the CC, can prevent the node from sending more than once in a TDMA round but fails to stop it from sending outside it scheduled transmission time (only if the CC's clock get faulty) where transmission slots are non-uniform. However, as mentioned in Section 4.3.2.3, each node is synchronising its clock with all other nodes using fault-tolerant average algorithm [59]. The BG has that same input from other nodes as well, and can synchronise its clock accordingly [8]. In the worst case, where the node's clock get faulty and the BG's clock has adjusted itself with the node's clock, this may result in a single bus access outside the node's scheduled time. This fault can be identified by using the membership service and such a node will be cut off from the rest of the cluster until it restarts itself in a healthy state and reintegrates into the cluster again. The BG must allow the CC to transmit at the correct time and should not block the full transmission of its frame within the correct transmission boundary as defined in the MEDL. Therefore, in order to tolerate the timing differences between the BG and the CC, the BG will open its gate to the bus a little earlier than the actual transmission time and closes a little later than the scheduled time [8] as shown in Figure 6.6. Further details are specified in the next section where slot timings of a transmitter, BG, and receiver nodes are formally verified.

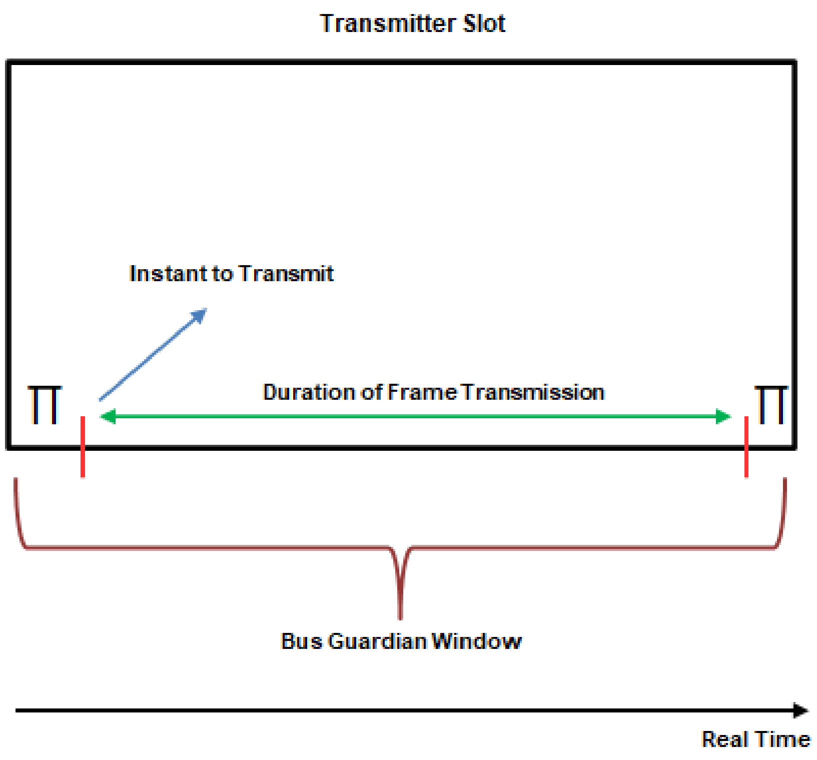

FIGURE 6.6: Bus Guardian Window

6.3 Formal verification of slot timing

Rushby [10] has published a formal verification for a TTA-based communication protocol, i.e. TTP. This thesis will build on Rushby's model for formal verification. However since Rushby's model only works for fixed and equal-length node slots in each TDMA round, there is a need to modify and extends this for my proposed approach. In INCUS+, the slot length of each node in each TDMA round is configured according to its transmission payload. Hence, the transmission slot length of a node can vary in different TDMA rounds. INCUS+ also supports dynamic communication schedules in different operational modes, which means the length of a cluster cycle can also vary in different operational modes. I have extended Rushby's model to formally verify the flexibility in slot window timings for each node in different TDMA rounds as well as in different operational modes.

This section will now formally verify the window timings of each slot of a node in different TDMA rounds of the cluster cycle, as well as in different operational modes. The basic pattern of communication is based on the global schedule, which holds the information about slot positions of transmitters, as well as slot start times and durations. A transmitter will start its frame transmission after some delay from its slot start time and finish the transmission some time before the allowed slot duration has elapsed. Similarly, a receiver starts listening for a frame at the beginning of its receive window time, and closes its receive window when the frame receive duration has expired. All the events are controlled at each node through its clock. Clocks in different nodes will deviate from each other but must be syncronised within a small threshold pertaining to the precision of clock synchronisation. As a consequence, if nodes did not account for that, it would be possible that a transmitter starts its frame transmission before some receiver starts listening for the frame, or finish its transmission when some receivers have already stopped listening for the frame. This situation would lead to inconsistencies among participating nodes, creating different cliques among nodes. Therefore, care must be taken while selecting parameters for window timings of transmitters, receivers, as well as bus guardians (responsible to block any transmission outside allocated time slot).

6.3.1 Parameters used for Formal Verification

The parameters used in the formal verification model are shown in Table 6.1. The values of these parameters, as shown in Figure 6.7, need to satisfy the following requirements:

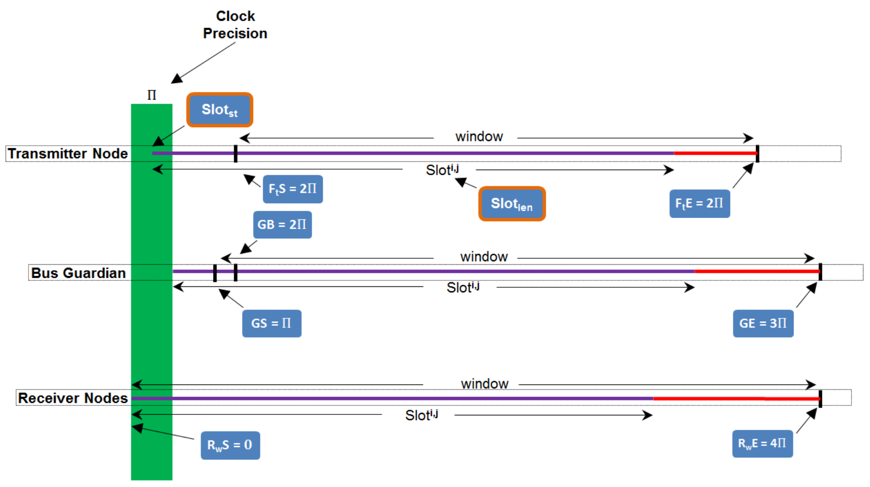

FIGURE 6.7: Slot Window Timing Parameters for formal verification extended from [10]

TABLE 6.1: Definition of the parameters used in the formal verification model

Parameter	**Description**
$Slot_{st}$	Time to start the node slot
$Slot_{len}$	Length of the node slot
F_tS	Maximum time to start frame transmission
F_tE	Time to end frame transmission
GS	Time to open bus guardian window
GB	Time to block the transmission
GE	Time to shut bus guardian window
R_wS	Time when receiver open its window to receive the frame
R_wE	Time when receiver closes its window

- **Agreement:** If a frame transmission over a non-faulty communication channel is received by any non-faulty receiver then all non-faulty receivers must receive the transmission.

- **Validation:** If a frame is transmitted by a non-faulty transmitter over a non-faulty communication channel then all non-faulty receivers must receive the transmitted frame.

6.3.2 Requirements and assumptions

Before starting the analysis, here is a recap of the requirements and assumptions of my INCUS+ model which are as follows:

1. INCUS+ does not use multiplexed slots, which means one slot per node in a TDMA round.
2. Slot duration ($Slot_{len}$) of a node is configured according to its transmission payload in each TDMA round.
3. One frame is transmitted per slot.
4. Each operational mode may have a different communication schedule, i.e. duration of TDMA rounds and Cluster Cycles can vary in different operational modes.

6.3.3 Hierarchy of communication

The hierarchy of communication is as follows:

Transmitter →Bus Guardian→Receivers

Thus the validity requirement is decomposed into two sub-requirements which are:

1. Frame transmission timing from the transmitter to its bus guardian.
2. Timing from the bus guardian to the receivers.

Here, we build on and extend Rushby's [10] model as follows:

The notion C is used for local clock time of each node. Uppercase letters are used for clock time quantities while lowercase letters are used to represent real-time quantities. Therefore we can say that $C_s^{i,j}(t)$ is the value of s's clock at real-time t in slot i of round j. For any participating node in a cluster, whether it is acting as a sender node s or as a receiver node r in any slot i of TDMA round j, their clocks are synchronised if the reading of the clocks of both nodes are within a precision Π. Therefore, according to clock synchronisation, we get:

Clock Sysnchronisation:

$$\left|C_s^{i,j}(t) - C_r^{i,j}(t)\right| \leq \Pi \tag{6.1}$$

The same needs to be true for any mode σ_i and therefore, we can say that for any participating node in a cluster, acting as a sender node s or as a receiver node r in any slot i of TDMA round j, their clocks are synchronised if the reading of the clocks of both nodes are within precision Π. Therefore, according to clock synchronisation, we get:

$$\left|C_s^{i,j}(t) - C_r^{i,j}(t)\right|_{\sigma_i} \leq \Pi \tag{6.2}$$

6.3.4 Requirement R1

If a frame is transmitted by a communication controller of a non-faulty transmitter then its non-faulty bus guardian must also allow the transmission.

6.3.4.1 Proof

Let us assume a node slot i start at $Slot_{st}^{i}$ where the length of the slot is exactly configured according to the transmission payload or frame size in that slot of a TDMA round. Therefore, we can say that the slot length for frame f in slot i of a TDMA round j should be $(Slot_{len}^{i,j}(f))$. The same will be the case for every mode σ_i. It is assumed that a transmitter starts its transmission at some offset F_tS after the start of the slot and ends the frame transmission at some offset F_tE after the time needed to transmit the frame. The associated BG also opens its window at some offset GS after the start of the slot and closes its window at some offset GE after the time needed to transmit the frame.

If s is a transmitter node at a point *t1* in real-time then the transmission start time for its frame f in its allocated slot i of TDMA round j will be:

$$C_s^{i,j}(t1) = Slot_{st}^{i,j}(f) + F_tS \tag{6.3}$$

For a mode σ_i the corresponding start time will be:

$$\left|C_s^{i,j}(t1)\right|_{\sigma_i} = \left|Slot_{st}^{i,j}(f) + F_tS\right|_{\sigma_i} \tag{6.4}$$

At this point, the BG for node s must already have opened its window in transmission slot i of round j to allow the transmitter to transmit its frame *f*, therefore, we need:

$$C_{bg}^{i,j}(t1) \geq Slot_{st}^{i,j}(f) + GS \tag{6.5}$$

Similarly, the BG for node s must already have opened its window in transmission slot i of round j in mode σ_i to allow the transmitter to transmit its frame *f*, therefore;

$$\left|C_{bg}^{i,j}(t1)\right|_{\sigma_i} \geq \left|Slot_{st}^{i,j}(f) + GS\right|_{\sigma_i} \tag{6.6}$$

Clocks of s and its BG must obey

$$-\Pi \leq C_{bg}^{i,j}(t1) - C_s^{i,j}(t1) \leq \Pi \tag{6.7}$$

By using Equation (6.3) we get

$$C_{bg}^{i,j}(t1) \geq Slot_{st}^{i,j}(f) + F_tS - \Pi \tag{6.8}$$

To satisfy Equation (6.5) we must have

$$F_tS \geq GS + \Pi \tag{6.9}$$

For mode σ_i, Clocks of s and its BG must obey

$$-\Pi \leq \left|C_{bg}^{i,j}(t1) - C_s^{i,j}(t1)\right|_{\sigma_i} \leq \Pi \tag{6.10}$$

By using Equation (6.4) we get

$$\left|C_{bg}^{i,j}(t1)\right|_{\sigma_i} \geq \left|Slot_{st}^{i,j}(f) + F_tS - \Pi\right|_{\sigma_i} \tag{6.11}$$

To satisfy Equation (6.6) we must have

$$|F_tS|_{\sigma_i} \geq |GS + \Pi|_{\sigma_i} \tag{6.12}$$

and this is clearly prove by the fact that parameters selected for window timing are $F_tS = 2\Pi$ and $GS = \Pi$.

Let us assume t2 is the physical point in time where s starts transmitting its frame ($F_tE \geq F_tS$); hence we can say that

$$C_{bg}^{i,j}(t2) = Slot_{st}^{i,j}(f) + Slot_{len}^{i,j}(f) + F_tE \tag{6.13}$$

At this point in time the transmitter BG window $C_{bg}^{i,j}(t2)$ must open, therefore

$$C_{bg}^{i,j}(t2) \leq Slot_{st}^{i,j}(f) + Slot_{len}^{i,j}(f) + GE \tag{6.14}$$

According to clock synchronisation this is within

$$-\Pi \leq C_{bg}^{i,j}(t2) - C_s^{i,j}(t2) \leq \Pi \tag{6.15}$$

Therefore, we can say that

$$C_{bg}^{i,j}(t2) \leq Slot_{st}^{i,j}(f) + Slot_{len}^{i,j}(f) + F_tE + \Pi \tag{6.16}$$

To satisfy Equation (6.16), $GE \geq F_tE + \Pi$ has to be satisfied. As parameters selected for window timings (see Figure 6.7) can illustrate that ($F_tE = F_tS$) therefore we can say that

$$GE \geq GS + 2\Pi \tag{6.17}$$

Equation (6.17) can clearly be proven by the fact that ($GS = \Pi$) and ($GE = 3\Pi$).

Now for a mode σ_i, let us assume t2 is the physical time where s is transmitting its frame ($F_tE \geq F_tS$); hence we can say that

$$\left|C_{bg}^{i,j}(t2)\right|_{\sigma_i} = \left|Slot_{st}^{i,j}(f) + Slot_{len}^{i,j}(f) + F_tE\right|_{\sigma_i} \tag{6.18}$$

At this point in time the corresponding BG window $\left|C_{bg}^{i,j}(t2)\right|_{\sigma_i}$ must be open, therefore it should be

$$\left|C_{bg}^{i,j}(t2)\right|_{\sigma_i} \leq \left|Slot_{st}^{i,j}(f) + Slot_{len}^{i,j}(f) + GE\right|_{\sigma_i} \tag{6.19}$$

According to clock synchronisation this is within

$$-\Pi \leq \left|C_{bg}^{i,j}(t2) - C_{s}^{i,j}(t2)\right|_{\sigma_i} \leq \Pi \tag{6.20}$$

Therefore, we can say that

$$\left|C_{bg}^{i,j}(t2)\right|_{\sigma_i} \leq \left|Slot_{st}^{i,j}(f) + Slot_{len}^{i,j}(f) + F_tE\right|_{\sigma_i} + \Pi \tag{6.21}$$

To satisfy Equation (6.21), $|GE|_{\sigma_i} \geq |F_tE|_{\sigma_i} + \Pi$ has to be satisfied. As parameters selected for window timings (see Figure 6.7) can satisfy that $(|F_tE|_{\sigma_i} = |F_tS|_{\sigma_i})$ therefore we can say that

$$|GE|_{\sigma_i} \geq |GS|_{\sigma_i} + 2\Pi \tag{6.22}$$

Equation (6.22) can clearly be proven by the fact that $(|GS|_{\sigma_i} = \Pi)$ and $(|GE|_{\sigma_i} = 3\Pi)$.

6.3.5 Requirement R2

If a non-faulty BG passes a frame then the frame will be received by all non-faulty receivers.

6.3.5.1 Proof

For a frame f of a transmitter in its node slot i of TDMA round j, let us assume the BG opens its window GS clock units after the slot start time for its receptive node and closes the window at $(Slot^{i,j}_{st}(f) + Slot^{i,j}_{len}(f) + GE)$. Let us assume receiver nodes are ready to listen for a frame at R_wS units after the slot start time of the transmitter node and stop receiving the frame at $(Slot^{i,j}_{st}(f) + Slot^{i,j}_{len}(f) + R_wE)$.

Let us assume at physical time t1:

$$C^{i,j}_{bg}(t1) = Slot^{i,j}_{st}(f) + GS \tag{6.23}$$

At time t1 when BG opens its window, the receiver r must already open its window to listen for a frame f, therefore, we can say that

$$C^{i,j}_{r}(t1) \geq Slot^{i,j}_{st}(f) + R_wS \tag{6.24}$$

For the receiver at instant t1 and by using the clock synchronisation rule, we satisfy

$$-\Pi \leq C_r^{i,j}(t1) - C_{bg}^{i,j}(t1) \leq \Pi \tag{6.25}$$

By using Equation (6.23), this transforms to

$$C_r^{i,j}(t1) \geq Slot_{st}^{i,j}(f) + GS - \Pi \tag{6.26}$$

and to prove Equation (6.24) we need $GS \geq R_w S + \Pi$ and this can be proved by substituting the values of parameters shown in Figure 6.7.

At any physical time t2 when the BG window is still open to allow the transmission such that

$$C_{bg}^{i,j}(t2) = Slot_{st}^{i,j}(f) + Slot_{len}^{i,j}(f) + GE \tag{6.27}$$

Therefore, the receiver window must be open at t2.

$$C_r^{i,j}(t2) \leq Slot_{st}^{i,j}(f) + Slot_{len}^{i,j}(f) + R_w E \tag{6.28}$$

According to clock synchronisation, it has

$$C_r^{i,j}(t2) \leq Slot_{st}^{i,j}(f) + Slot_{len}^{i,j}(f) + GE + \Pi \tag{6.29}$$

To validate Equation (6.28), we need $R_w E \geq GE + \Pi$ and this can be proven by substituting the values of parameters GE and $R_w E$.

Let us assume the current operational mode is σ_i. For a frame f of a transmitter in its node slot i of TDMA round j, the BG opens its window GS clock units after the slot start time for its respective node and closes the window at

$(\left|Slot^{i,j}_{st}(f) + Slot^{i,j}_{len}(f) + GE\right|_{\sigma_i})$. Receiver nodes are ready to listen for a frame at R_wS units after the slot start time of the transmitter node and stop receiving the frame at $(\left|Slot^{i,j}_{st}(f) + Slot^{i,j}_{len}(f) + R_wE\right|_{\sigma_i})$.

Let us assume at physical time t1:

$$\left|C^{i,j}_{bg}(t1)\right|_{\sigma_i} = \left|Slot^{i,j}_{st}(f) + GS\right|_{\sigma_i} \tag{6.30}$$

At time t1 when the BG opens its window, the receiver r must already open its window to listen for a frame *f*, therefore, we can say that

$$\left|C^{i,j}_{r}(t1)\right|_{\sigma_i} \geq \left|Slot^{i,j}_{st}(f) + R_wS\right|_{\sigma_i} \tag{6.31}$$

For the receiver at instant t1 and by using the clock synchronisation rule we satisfy

$$-\Pi \leq \left|C^{i,j}_{r}(t1) - C^{i,j}_{bg}(t1)\right|_{\sigma_i} \leq \Pi \tag{6.32}$$

By using Equation (6.30), we get

$$\left|C^{i,j}_{r}(t1)\right|_{\sigma_i} \geq \left|Slot^{i,j}_{st}(f) + GS\right|_{\sigma_i} - \Pi \tag{6.33}$$

and to prove Equation (6.31) we need $GS \geq R_wS + \Pi$ and this can be proved by substituting the values of parameters shown in Figure 6.7.

At any physical time t2 when the BG window is still open to allow the transmission such that

$$\left|C^{i,j}_{bg}(t2)\right|_{\sigma_i} = \left|Slot^{i,j}_{st}(f) + Slot^{i,j}_{len}(f) + GE\right|_{\sigma_i} \tag{6.34}$$

Therefore, the receiver window must be open at t2.

$$\left|C_r^{i,j}(t2)\right|_{\sigma_i} \leq \left|Slot_{st}^{i,j}(f) + Slot_{len}^{i,j}(f) + R_w E\right|_{\sigma_i} \tag{6.35}$$

According to clock synchronisation, we get

$$\left|C_r^{i,j}(t2)\right|_{\sigma_i} \leq \left|Slot_{st}^{i,j}(f) + Slot_{len}^{i,j}(f) + GE\right|_{\sigma_i} + \Pi \tag{6.36}$$

To validate Equation (6.35), we need $R_w E \geq GE + \Pi$ and this can be proven by substituting the values of parameters GE and $R_w E$.

The validation property is clearly proven by the requirements *R1* and *R2*. The first requirement (R1) ensures that a frame transmitted by a non-faulty transmitter will be passed by its non-faulty BG whereas the second requirement (R2) makes sure that a frame passed by a non-faulty BG should be received by all non-faulty receivers.

For the agreement property, the *R2* proves that a frame transmitted by a non-faulty transmitter must be passed by its non-faulty BG, which ensures that all non-faulty receivers will receive the frame. If this is taken in other way round, where a faulty transmitter is trying to transmit a frame then it may attempt to transmit the frame at an incorrect time. If so, then its BG will block such a transmission, if it falls entirely outside the allowed transmission window, or it will truncate the frame if it falls partially outside the transmission window. Therefore, if a transmission was blocked by a BG then it will not be observed by any receiver. Similarly, if a transmission is truncated by a BG, then all the non-faulty receivers will reject such a transmission.

6.3.6 Prevention of slot overlapping

So far, the model assures that a non-faulty transmitter must transmit its frame within its slot boundaries and its non-faulty BG must allow that transmission. Another issue that needs to be tackled is overlapping slots. A frame transmitted by a non-faulty transmitter must not interfere with the next transmitter opening its transmission window.

The design rule to prevent such erroneous scenarios [10] is:

- The next transmission takes place no earlier than 4Π after the end of the previous transmission.

Please note that INCUS+ is using a slot length configuration of a node on the basis of its payload in different TDMA rounds as well as in different operational modes, but we still only transmit one frame per slot. If a frame f is transmitted in slot i of round j then the next slot should be $(i{+}1,j)$. Therefore, the above design rule can be formalised in INCUS+ as follows:

$$Slot_{st}^{i+1,j}(f) \geq Slot_{st}^{i,j}(f) + Slot_{len}^{i,j}(f) + 4\Pi \tag{6.37}$$

Similarly for a mode σ_i, we get:

$$\left|Slot_{st}^{i+1,j}(f)\right|_{\sigma_i} \geq \left|Slot_{st}^{i,j}(f) + Slot_{len}^{i,j}(f)\right|_{\sigma_i} + 4\Pi \tag{6.38}$$

6.3.7 Requirement R3

The window of one communication controller must not overlap with the window of the next communication controller in the subsequent slot. Therefore, we can say that a non-faulty communication controller of a transmitter node must finish its transmission before the next node's non-faulty BG opens its window for the next slot.

6.3.7.1 Proof

Let us assume at real time t, a communication controller of a transmitter node s ends its transmission for a frame f in its slot i of round j, therefore

$$C_s^{i,j}(t) = Slot_{st}^{i,j}(f) + Slot_{len}^{i,j}(f) + F_tE \tag{6.39}$$

Now, node s's bus guardian C_{bg} window should be open for the next slot $(i,j{+}1)$ no earlier than t with

$$C_{bg}^{i,j+1}(t) \leq Slot_{st}^{i,j+1}(f) + GS \tag{6.40}$$

As clock synchronisation requires

$$-\Pi \leq C_{bg}^{i,j+1}(t) - C_s^{i,j}(t) \leq \Pi \tag{6.41}$$

By using Equation (6.39), we get

$$C_{bg}^{i,j+1}(t) \leq Slot_{st}^{i,j}(f) + Slot_{len}^{i,j}(f) + F_tE + \Pi \tag{6.42}$$

To satisfy Equation (6.40), we need

$$Slot_{st}^{i,j+1}(f) \geq Slot_{st}^{i,j}(f) + Slot_{len}^{i,j}(f) + F_tE + \Pi - GS \tag{6.43}$$

This satisfies Equation (6.37) by using the values of $F_tE = 2\Pi$ and $GS = \Pi$

Let us assume current operational mode is σ_i, and at real time t, a transmitter node s ends its transmission for a frame f in its slot i of round j, therefore

$$\left|C_s^{i,j}(t)\right|_{\sigma_i} = \left|Slot_{st}^{i,j}(f) + Slot_{len}^{i,j}(f) + F_tE\right|_{\sigma_i} \tag{6.44}$$

Now, that node's bus guardian C_{bg} window should be open for the next slot (*i,j+1*) in the same operational mode no earlier than t

$$\left|C_{bg}^{i,j+1}(t)\right|_{\sigma_i} \leq \left|Slot_{st}^{i,j+1}(f) + GS\right|_{\sigma_i} \tag{6.45}$$

As clock synchronisation requires

$$-\Pi \leq \left|C_{bg}^{i,j+1}(t) - C_{s}^{i,j}(t)\right|_{\sigma_i} \leq \Pi \tag{6.46}$$

By using Equation (6.44), we get

$$\left|C_{bg}^{i,j+1}(t)\right|_{\sigma_i} \leq \left|Slot_{st}^{i,j}(f) + Slot_{len}^{i,j}(f) + F_tE\right|_{\sigma_i} + \Pi \tag{6.47}$$

To satisfy Equation (6.45), we need

$$\left|Slot_{st}^{i,j+1}(f)\right|_{\sigma_i} \geq \left|Slot_{st}^{i,j}(f) + Slot_{len}^{i,j}(f) + F_tE\right|_{\sigma_i} + \Pi - |GS|_{\sigma_i} \tag{6.48}$$

This satisfies Equation (6.38) by using the values of $F_tE = 2\Pi$ and $GS = \Pi$

Slot overlapping may also occur because of a faulty BG, that may result in a potential transmission collision. The design rule to prevent such issues in INCUS+ is as follows:

- No transmission is allowed if a transmitter does not start its transmission within the boundary of $Slot_{st}^{i,j}(f) + F_tS$ within an operational mode.

As INCUS+ now uses variable slot lengths for each node in each TDMA round of an operational mode, the above requirement *R3* to prevent slot overlapping would not be sufficient for all scenarios such as a BG and the transmitting node agreeing on their slot positioning, but assumed to be in different TDMA rounds. The bus guardian parameter GB serves to overcome such an issue. To this end, we have the following requirement.

6.3.8 Requirement R4

If a transmitter node is trying to transmit its frame in the allocated slot, but after its F_tS has elapsed, then its bus guardian must block such a transmission.

6.3.8.1 Proof

Let us suppose s is a transmitter node in slot i of round j to transmit its frame f at physical time t where F_tS has already elapsed, therefore

$$C_s^{i,j}(t) > Slot_{st}^{i,j}(f) + F_tS \tag{6.49}$$

At this point, the bus guardian for s must already block the controller from transmission. Therefore

$$C_{bg}^{i,j}(t) = Slot_{st}^{i,j}(f) + GB \tag{6.50}$$

Again, clock synchronisation requires

$$-\Pi \leq C_{bg}^{i,j}(t) - C_s^{i,j}(t) \leq \Pi \tag{6.51}$$

Therefore, we can say that

$$C_{bg}^{i,j}(t) < Slot_{st}^{i,j}(f) + F_tS + \Pi \tag{6.52}$$

To satisfy Equation (6.50), we must have

$$GB < F_tS + \Pi \tag{6.53}$$

and this is clearly proven by the fact that $GB = 2\Pi$ and $F_tS = 2\Pi$.

For an operational mode σ_i, let us assume s is a transmitter node in slot i of round j to transmit its frame f at physical time t where F_tS has already elapsed, therefore

$$\left|C_s^{i,j}(t)\right|_{\sigma_i} > \left|Slot_{st}^{i,j}(f) + F_tS\right|_{\sigma_i} \tag{6.54}$$

At this point, bus guardian for s must already block the controller from transmission. Therefore

$$\left|C_{bg}^{i,j}(t)\right|_{\sigma_i} = \left|Slot_{st}^{i,j}(f) + GB\right|_{\sigma_i} \tag{6.55}$$

Again, clock synchronisation requires

$$-\Pi \leq \left|C_{bg}^{i,j}(t) - C_s^{i,j}(t)\right|_{\sigma_i} \leq \Pi \tag{6.56}$$

Therefore, we can say that

$$\left|C_{bg}^{i,j}(t)\right|_{\sigma_i} < \left|Slot_{st}^{i,j}(f) + F_tS\right|_{\sigma_i} + \Pi \tag{6.57}$$

To satisfy Equation (6.55), we must have

$$|GB|_{\sigma_i} < |F_tS|_{\sigma_i} + \Pi \tag{6.58}$$

and this is clearly proven by the fact that $|GB|_{\sigma_i} = 2\Pi$ and $|F_tS|_{\sigma_i} = 2\Pi$.

6.4 Protocol Behaviour under Faults

In the formal verification model, the timing constraints are proven by assuming that all the participating nodes (transmitters, receivers, and bus guardians) are non-faulty. For example, a frame transmitted by a non-faulty node should be passed through its non-faulty bus guardian. Now, let us analyse different fault scenarios and examine the behaviour of INCUS+ to detect such errors in both the time and value domains. Please note that as with all TTA-based Safety-Critical Real-Time (SCRT) protocols, the fault hypothesis is to handle a single fault at a time.

6.4.1 What if the transmitter fails?

There are multiple reasons for a transmitter to fail and this section will discuss multiple failure scenarios and protocol behaviour against these scenarios. As we have to guarantee fail-silence, we will focus on fail-silence violations.

6.4.1.1 Completely off the scheduled frame transmission

It is quite possible for a faulty node to transmit in an interval that is not scheduled for its frame transmission. In this case it will violate Equation (6.3) of *R1* which requires:

$$C_s^{i,j}(t1) = Slot_{st}^{i,j}(f) + F_tS \tag{6.59}$$

This will be handled by Equation (6.5), that is:

$$C_{bg}^{i,j}(t1) \geq Slot_{st}^{i,j}(f) + GS \tag{6.60}$$

As the bus guardian has a copy of the MEDL, it exactly knows the transmission scheduled for its node. The bus guardian window will remain closed outside these intervals therefore any such transmission attempt by the associated node will fail and no other node will be able to receive such a transmission, or have their own transmission interfered with.

6.4.1.2 Transmitter transmitting longer than expected

A faulty node may try to occupy the communication channel longer than its allocated slot length. By the end of the allocated slot, the bus guardian will already shut it window such that:

$$C_{bg}^{i,j}(t) \geq Slot_{st}^{i,j}(f) + Slot_{len}^{i,j}(f) + GE \tag{6.61}$$

Therefore, such a transmission will be truncated by the bus guardian (as the bus guardian window will be closed after the allowed transmission time (defined in MEDL) has been elapsed). The resulting, truncated frame still be received by the receivers, however, this incorrect frame will be detected by other protocol services in the value domain. In particular, the CRC check on the received frame will fail at non-faulty receivers. As a consequence, all the non-faulty receivers will discard such a frame and they will remove the transmitter from their membership list.

6.4.1.3 Transmission slot position is incorrect

Another fault scenario may arise where the transmitter node assumes itself in a different slot than the actual slot. For example, the actual slot position of a node is slot i of round j, but the node is considering itself in a slot i of round *j+1*. Unlike the existing TTA-based approaches, the slot length of the node in INCUS+ may vary in different TDMA rounds, therefore, further failure scenarios may arise out of this.

1. The slot length of a node in round *j+1* is less than the slot length in round j i.e. ($Slot_{len}^{i,j}(f) > Slot_{len}^{i,j+1}(f)$). In this case, the bus guardian window, which is greater than its window in round *j+1* will allow the transmitter to to transmit its frame. While that means, in the time domain, this error will not be detected, at receivers, this error will be detected in the value domain due to the Cyclic Redundancy Check (CRC) checksum of the Controller State (CState) for the transmitter being different from the rest of the nodes.

Therefore, such frames will be rejected by the receivers and the transmitter node will not be acknowledged in subsequent slots. This will cause transmitter to lose its membership, restart after one round, and reintegrate with the cluster.

2. The slot length of a node in round *j+1* is greater than the slot length in round j (i.e. $Slot_{len}^{i,j}(f) < Slot_{len}^{i,j+1}(f)$). In this case, the bus guardian window, which is less than its window in round *j+1* will truncate the frame as the transmitter will try to transmit outside the allowed duration. At this instant, bus guardian will shut its window

$$C_{bg}^{i,j}(t) \geq Slot_{st}^{i,j}(f) + Slot_{len}^{i,j}(f) + GE \tag{6.62}$$

Therefore, such transmissions will be truncated by the bus guardian (as the bus guardian window will be shut after the actual transmission time defined in the MEDL for slot i of round j has elapsed). This truncated frame will still be received by the receivers, however, truncated frames are already rejected by other protocol services in the value domain, such as the CRC check. The CRC check on the received frame will fail, and as a consequence, all non-faulty receivers will discard such a frame and will remove the transmitter from their membership list.

6.4.1.4 Transmitter assumes itself in a completely different operational mode

Another fault scenario may arise where the transmitter node assumes itself in a different operational mode than the actual mode. For example, the actual slot position of a node is slot i of round j in an operational mode σ_i while the node considers itself in slot i of round j of operational mode σ_{i+1}. Unlike the existing TTA-based approaches, the slot length of the node in INCUS+ may vary in different TDMA rounds as well as in different operational modes, therefore, further failure scenarios may arise out of this.

1. The length of a slot i of a node in round j of an operational mode σ_{i+1} is less than the slot length i in round j of an operational mode σ_i i.e. $(\left|Slot_{len}^{i,j}(f)\right|_{\sigma_i} > \left|Slot_{len}^{i,j}(f)\right|_{\sigma_{i+1}})$. In this case, the bus guardian window, which is greater than its window in slot i of round j in mode σ_{i+1} will allow the transmitter to to transmit its frame. While that means, in the time domain, this error will not be detected, at receivers, this error will be detected in the value domain due to the Cyclic Redundancy Check (CRC) checksum of the Controller State (CState) for the transmitter being different from the rest of the nodes. Therefore, such frames will be rejected by the receivers and the transmitter node will not be acknowledged in subsequent slots of mode σ_i. This will cause the transmitter to lose its membership, restart after one round, and reintegrate with the cluster.

2. The length of a slot i of a node in round j of an operational mode σ_{i+1} is greater than the slot length i in round j of an operational mode σ_i, i.e. $(\left|Slot_{len}^{i,j}(f)\right|_{\sigma_i} < \left|Slot_{len}^{i,j}(f)\right|_{\sigma_{i+1}})$. In this case, the bus guardian window, which is less than its window in slot i of round j in operational mode σ_{i+1} will truncate the frame as the transmitter will try to transmit outside the allowed duration. At this instant, the bus guardian will shut its window

$$\left|C_{bg}^{i,j}(t)\right|_{\sigma_i} \geq \left|Slot_{st}^{i,j}(f) + Slot_{len}^{i,j}(f) + GE\right|_{\sigma_i} \tag{6.63}$$

Therefore, such transmissions will be truncated by the bus guardian (as the bus guardian window will be shut after the actual transmission time defined in the MEDL for slot i of round j in operational mode σ_i has elapsed). This truncated frame will still be received by the receivers, however, truncated frames are already rejected by other protocol services in the value domain, such as the CRC check. The CRC check on the received frame will fail, and as a consequence, all non-faulty receivers will discard such a frame and will remove the transmitter from their membership list.

6.4.2 What if the Bus Guardian fails?

In the previous scenario, a faulty transmitter is discussed and now, we will discuss what happens if a bus guardian fails. There are multiple failure scenarios for a bus guardian and each will be discussed one by one in the remainder of this section.

6.4.2.1 Bus guardian blocks correct transmission

If a bus guardian gets faulty and blocks the correct transmission from its respective transmitter node. This will violate Equation (6.23) of *R2* which is:

$$C_{bg}^{i,j}(t1) = Slot_{st}^{i,j}(f) + GS \tag{6.64}$$

In this case none of the receivers will receive the expected frame. Therefore, in the value domain this will be detected by the membership service and all the non-faulty receivers will remove the transmitter from their membership vector, which will force the node to restart and attempt to reintegrate.

6.4.2.2 The Bus guardian truncates a correct transmission

If the bus guardian gets faulty in such a way that it truncates a correct frame, then in the value domain, again the CRC checksum at receivers will be detected as incorrect and all the receivers will discard such a frame and remove the transmitter from their membership list. The transmitter will detect such an error when it finds its CState not being consistent with the majority of the nodes in the cluster.

6.4.2.3 The slot positioning for a Bus guardian is incorrect

A fault scenario may arise when a bus guardian assume itself in a different slot than the actual slot. For example, the actual slot position of a guardian is slot i of round j while the bus guardian is considering itself in a slot i of round *j+1*. Unlike the existing TTA based approaches, the slot length of the node in INCUS+ may vary in different TDMA rounds and so would the bus guardian window. Therefore, further failure scenarios may arise out of this.

1. The slot length of a node in round *j+1* is less than the slot length in round *j*. In this case, the bus guardian window which should be less than its window in round *j* will be truncating the correct frame, which again violates *R2*. Therefore, the CRC check on the receiving frame will fail at non-faulty receivers. As a consequence, all the non-faulty receivers will discard such a frame and will remove the transmitter from their membership list. This then will force the transmitter to restart and attempt to reintegrate into the cluster.

2. The slot length of a node in round *j* is less than the slot length in round *j+1*, the bus guardian window, which should be greater than its window in round *j* will allow the transmitter to transmit its frame (if the transmitter starts transmission within the boundary of $Slot_{st}^{i,j}(f) + F_tS$). This error will be detected in subsequent rounds where the bus guardian window starts truncating the correct frame when its window opens for less time than the required time to transmit a frame, as per the previous scenario. In the worst case scenario, it may take up to a full cluster cycle time to detect such an error.

3. The slot length of a node in round *j* is less than the slot length in round *j+1*, the bus guardian window would be greater than its window in round *j*, which will again allow the transmitter to to transmit its frame. As the bus guardian window would be open longer than it should be, frame transmission could be thought to overlap with the next slot (if transmission starts after $Slot_{st}^{i,j}(f)+F_tS$). But this would violate the *R4*, which is:

$$C_s^{i,j}(t) \leq Slot_{st}^{i,j}(f) + F_tS \tag{6.65}$$

Therefore, the bus guardian will shut its window at this instant such that:

$$C_{bg}^{i,j}(t) = Slot_{st}^{i,j}(f) + GB \tag{6.66}$$

This means the bus guardian will block any transmission that starts after $Slot_{st}^{i,j}(f) + F_tS$.

6.4.2.4 The Bus guardian assumes itself in a completely different operational mode

A fault scenario may arise when a bus guardian assume itself in a different operational mode than the actual mode. For example, the actual slot position of a guardian is slot i of round j in mode σ_i while the bus guardian is considering itself in a slot i of round j in mode σ_{i+1}. Unlike the existing TTA-based approaches, the slot length of the node in INCUS+ may vary in different operational modes and so does the bus guardian window. Therefore, further failure scenarios may arise out of this.

1. Lets say the slot length i of a node in round j during an operational mode σ_{i+1} is less than the slot length i in round j during the operational mode σ_i . In this case, the bus guardian window, which should be less than its window in mode σ_i will be truncating the correct frame, which again violates *R2*. Therefore, the CRC check on the receiving frame will fail at non-faulty receivers. As a consequence, all the non-faulty receivers will discard such a frame and will remove the transmitter from their membership list. This then will force the transmitter to restart and attempt to reintegrate into the cluster.

2. If the slot length i of a node in round j during the operational mode σ_i is less than the slot length i in round j during the operational mode σ_{i+1}, the bus guardian window which should be greater than its window in mode σ_i will allow the transmitter to transmit its frame (if the transmitter starts

transmission within the boundary of $\left|Slot_{st}^{i,j}(f) + F_tS\right|_{\sigma_i}$). This error will be detected in subsequent rounds where bus guardian window starts truncating the correct frame when its window opens for less time than the required time to transmit a frame, as per the previous scenario. In the worst case scenario, it may take up to a full cluster cycle time to detect such an error.

3. If the slot length i of a node in round j during the operational mode σ_i is less than the slot length i in round j during the operational mode σ_{i+1}, the bus guardian window would be greater than its window in mode σ_i, which will again allow the transmitter to transmit its frame. As the bus guardian window would be open longer than it should be, the frame transmission can be considered to overlap with the next slot (if transmission starts after $\left|Slot_{st}^{i,j}(f) + F_tS\right|_{\sigma_i}$). But this will violate *R4*, which is:

$$\left|C_{s}^{i,j}(t)\right|_{\sigma_i} \leq \left|Slot_{st}^{i,j}(f) + F_tS\right|_{\sigma_i} \tag{6.67}$$

   Therefore, the bus guardian will shut its window at this instant such that:

$$\left|C_{bg}^{i,j}(t)\right|_{\sigma_i} = \left|Slot_{st}^{i,j}(f) + GB\right|_{\sigma_i} \tag{6.68}$$

   This means the bus guardian will block any transmission that starts after $\left|Slot_{st}^{i,j}(f) + F_tS\right|_{\sigma_i}$.

6.5 Summary

This chapter presented the enforcement of fail-silent behaviour of INCUS+ using a special independent device known as a bus guardian. A bus guardian is added to each node to protect the communication channel from a node's transmission at arbitrary points in time. According to my proposed protocol, each node is allowed to access the communication channel at pre-defined points in time which are held by a data structure called the MEDL. Different types of bus guardians were presented in this chapter, but in order to support the flexibility of INCUS+ which allows dynamic communication schedules with variable slot lengths in different TDMA rounds as well as in different operational modes, only independent bus guardians with the capability of listening to full incoming traffic will offer a solution. This may be a more expensive approach in a bus topology with full redundancy, but should be a feasible approach if guardians are used as star couplers. In a practical application, I would expect the additional costs to be more than offset by the additional efficacy of bus utilisation. A formal verification model was also used in this chapter to verify the adherence to fail-silent behaviour of the INCUS+ in the time domain. The timing parameters for a transmitter, receiver, and bus guardian were formally verified in the time domain. The bahaviour of the protocol was analysed under different faults. We saw that the proposed approach can handle most faults in the time domain, and that those faults that cannot be detected in the time domain are tackled in the value domain by using protocol services such as the CRC check and the membership service.

Chapter 7

Conclusion and Future Work

This thesis explored how flexibility needed for today's application can be achieved for safety-critical real-time systems. Two major existing TTA-based communication protocols were TTP/C and FlexRay. These protocols are widely used in safety-critical real-time systems as they are tilted heavily towards the reliability required for such dependable systems. The big disadvantage of TTA-based protocols is the lack of flexibility. They do not support dynamic communication schedules for safety-critical information. In this thesis, I have presented the need of flexible communication schedules for safety-critical real-time systems through representative case studies such as brake-by-wire systems and autonomous vehicles. It was argued that the functionality of each node connected in such systems is different and so are their payload requirements. Therefore, the existing systems that assign equal-length time slots to all nodes result in poor channel utilisation. This severely impacts achievable channel utilisation for safety-critical real-time systems. Here, I have presented an iterative approach to overcome these deficiencies in a systematic way. As a first iteration in my approach, INCUS was designed as a communication protocol that uses a flexible communication schedule (time slots) for each node based on the node's transmission requirements. Thus, I was able to achieve a length of a time slot for each node that was different from that of the other nodes inside a TDMA round. My protocol, INCUS, achieves almost twofold the transmission efficiency in a typical automotive brake-by-wire system scenario.

In subsequent design iterations, I was able to expand the span of communication schedules over a number of TDMA rounds, leading towards an even more flexible approach called INCUS+. For validation, a number of use-cases such as Advanced Driver Assistance Systems (ADAS) in an autonomous vehicle, Unmanned Aerial Vehicles (UAVs), and healthcare systems based on Internet-of-Things (IoT) were investigated and shown to exhibit the need for different transmission slot lengths of a node during different TDMA rounds of a cluster cycle. This demonstrated that the length of TDMA rounds can also vary during an entire cluster cycle (over multiple TDMA rounds). INCUS+ significantly improved channel utilisation over TTA-based protocols, while guaranteeing atomicity and all required safety features at the protocol level. Slot length configurations of each node in accordance with its actual transmission payload requirements in each TDMA round of a cluster cycle eliminated node slot idle times for all nodes and hence, significantly reduced transmission overhead time. Compared to FlexRay, this significantly improved bandwidth utilisation. In my analysis, I have shown that this kind of flexibility makes it possible to reduce the gross overhead time by almost 99%, improving overall bandwidth utilisation efficiency almost nine times compared to FlexRay in an autonomous vehicle system case study. Furthermore, flexibility was enhanced by allowing different operational modes with different communication schedules. This additional flexibility was achieved by switching between different operational modes at run time.

A crucial next step was to ensure the same level of dependability is maintained with this gained level of flexibility, most notably the prevention of a single point of failure and to guarantee fail-silence. To this end, the bus guardian mechanism [181] had to be extended to prevent the bus from monopolisation of a sending node that was trying to transmit outside its allocated transmission windows. The bus guardian mechanism of existing TTA-based communication protocols [7, 33, 80, 81, 92] were not capable of handling the flexible communication schedules introduced here, as it was designed to work with static and equal-length time slots for all nodes. I, therefore designed an independent bus guardian approach that has full knowledge of the transmission schedules and actively listens to the full traffic over the communication channel. This made it possible to handle babbling idiot-faults with my flexible communication schedules. This thesis also used formal verification to ensure the correctness of the proposed protocol and its bus guardians. Protocol behaviour under different faults was analysed and verified by using a formal verification model. Under the fault hypothesis, the transmitters and bus guardian nodes were analysed and my formal verification showed that the proposed protocol along with its bus guardians can successfully handle all of these faults.

Next, I needed to investigate the emergent complexity when implementing my flexible protocol for today's applications. The existing protocols for safety-critical real-time systems such as TTP/C [33] and FlexRay [7] were designed at a low, procedural level. A key disadvantage of such an implementation was the design and development effort, which can take up to several man-years. The rigorous timing requirements that need to be modelled early on in the design process has made it difficult to model verifiable executable real-time behaviour at a high level. When designing my initial version of INCUS+ without using a subsumption architecture, one pattern that become apparent in this initial design was a replication of concerns. This lead me to a refined modelling approach that greatly enhances the modularity of the design. I was able to show that a high-level implementation of a communication protocol for safety-critical real-time systems based on the subsumption architecture was not only possible, but facilitates the incremental development of the system using executable models throughout. The modelling technique presented in this thesis allows designing flexibility into communication protocols while retaining the strict predictability and timing required by dependable real-time systems. This ensures the development of a communication subsystem without losing the fundamental properties of predictable real-time performance. The subsumption architecture made it possible to incrementally refine the implementation by adding, modifying, or changing the behaviour of a sub-system without interfering with unaffected components of the system.

7.1 Future Work

This thesis has provided flexible communication mechanisms for safety-critical real-time systems. A useful addition in the proposed research work would be to use the same communication medium such as bus to transmit both safety-critical and non safety-critical information. FlexRay used the same channel for safety-critical and non safety-critical information however, they implemented fixed length communication schedules for safety-critical part of information. The key challenge to design and implement such an approach with flexible communication schedules will be to guarantee the timeliness for safety-critical information while maintaining the same level of fault-tolerance (used in proposed approach) at protocol level. Another interesting idea will be the design and implementation of a gateway node that can be used as an interface between safety-critical and non-safety-critical real-time communication. Designing a system completely as a safety-critical system without the requirement of safety-critical services in every step may lead to an inflexible system and an expensive one too. A consequence of this part of research will be to provide a gateway node between safety-critical and non-safety-critical real-time systems. Communication protocols from both domains exist but they are not capable of operating in a hybrid environment because they are designed to operate in a homogenous environment only. Functionality of communication protocols, such as Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) that can be used for non-safety-critical systems is not providing any support for safety-critical real-time systems. But combining both kind of systems through an gateway node can have the advantage of resolving real world problems. The role of a gateway node will be to firewall the safety-critical information from non safety-critical information. The key challenges for a gateway node is to ensure the timeliness for safety-critical part of the communication. For this, the gateway node must be aware of the complete transmission schedule of safety-critical information so that it can block any transmission from non safety-critical real-time systems that can interrupt the communication from safety-critical real-time systems. Other challenges may include reliable delivery of safety-critical information in-case of failure of the gateway node. The gateway idea can further be extended to develop Intelligent Transportation Systems. It can be used to link safety-critical clusters of a wired network with a wireless network such as V2X (Vehicle to everything) architecture to exchange the information on road infrastructure, congestion, and the like.

Appendix A

List of Publications

Parts of this thesis have been accepted and published in the following International Conferences and Journals.

1. F. R. Raja, D. Chen, R. Hexel, "A Flexible Communication Protocol with Guaranteed Determinism for Distributed, Safety-Critical Real-Time Systems," IEEE Access (10) 2022, pp. 48049-48070.

   My Contribution: Design and Implementation of INCUS+ and literature review. The part of this work is presented in Chapter 4 and Chapter 6.

2. D. Chen, R. Hexel, and F. R. Raja, "Engineering real-time communication through time-triggered subsumption," in Proceedings of the 11th International Conference on Evaluation of Novel Software Approaches to Software Engineering. ENASE, 2016, pp. 272–281.

   My Contribution: Design and Implementation of the protocol using LLFSMs and literature review. The part of this work is presented in Chapter 5.

3. D. Chen, R. Hexel, and F. R. Raja, "INCUS: A communication protocol for safety critical distributed real time systems," in The 20th Asia-Pacific Conference on Communication (APCC2014). IEEE, 2014, pp. 309–314.

   My Contribution: Design and Implementation of INCUS and literature review. The part of this work is prsented in Chapter 4.

References

[1] "Real time systems." http://retis.sssup.it/~giorgio/rts-MECS.html. Accessed: 2021-09-09.

[2] H. Kopetz, *Real-time systems: design principles for distributed embedded applications*. Springer, 2011.

[3] R. Obermaisser, *Time-Triggered Communication*. CRC Press, 2011.

[4] H. Kopetz, "A communication infrastructure for a fault-tolerant distributed real-time system," *Control Engineering Practice*, vol. 3, no. 8, pp. 1139–1146, 1995.

[5] H. Kopetz, "The time-triggered model of computation," in *Proceedings 19th IEEE Real-Time Systems Symposium (Cat. No. 98CB36279)*, pp. 168–177, IEEE, 1998.

[6] C. Scheidler, G. Heiner, R. Sasse, E. Fuchs, H. Kopetz, and C. Temple, "Time-triggered architecture (TTA)," *Advances in Information Technologies: The Business Challenge*, pp. 758–765, 1997.

[7] F. Consortium *et al.*, "FlexRay communications system-protocol specification," *Version*, vol. 2, no. 1, pp. 198–207, 2005.

[8] "Time-triggered protocol TTP/C high-level specification, document protocol version 1.1, TTTech document number d-032-s-10-028," 2004.

[9] S. Ramberger, W. Herzner, E. Schoitsch, and W. Kubinger, "Ttipp3-a fault-tolerant time-triggered platooning demonstrator," in *Intelligent Solutions in Embedded Systems, 2008 International Workshop on*, pp. 1–11, IEEE, 2008.

[10] J. Rushby, "Formal verification of transmission window timing for the time-triggered architecture," *Technical Report Deliverable 24b, SRI Pro... ject 11003*, 2001.

[11] K. Juvva, "Real-time systems," *Topics in Dependable Embedded Systems*, no. 28, 1998.

[12] H. Kopetz, "Event-triggered versus time-triggered real-time systems," in *Operating Systems of the 90s and Beyond*, pp. 86–101, Springer, 1991.

[13] *ANSI / IEEE Carrier Sense Multiple Access with Collision Detection (CS-MA/CD) Access Method and Physical Layer Specification.* 345 East 47th Street, New York NY 10017: The Institute of Electrical and Electronic Engineering, Inc., 1985.

[14] I. . W. Group *et al.*, "Ieee standard for information technology–telecommunications and information exchange between systems local and metropolitan area networks–specific requirements part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications," *IEEE Std 802.11-2012 (Revision of IEEE Std 802.11-2007)*, pp. 1–2793, 2012.

[15] H. Kopetz, R. Hexel, A. Krüger, D. Millinger, R. Nossal, A. Steininger, C. Temple, T. Führer, R. Pallierer, and M. Krug, "A prototype implementation of a TTP/C controller," in *Proc. of the SAE Congress 1997*, (Detroit, MI, USA), Society of Automotive Engineers, SAE Press, Feb. 1997. SAE Paper No. 970296.

[16] V. Estivill-Castro and R. Hexel, "Simple, not simplistic — the middleware of behaviour models," in *ENASE 10 International Conference on Evaluation of Novel Approaches to Software Engineering*, (Barcelona, Spain), INSTCC, April 2015.

[17] N. Navet, Y.-Q. Song, and F. Simonot, "Worst-case deadline failure probability in real-time applications distributed over controller area network," *Journal of systems Architecture*, vol. 46, no. 7, pp. 607–617, 2000.

[18] SAE, "Glossary of vehicle networks for multiplexing and data communications, SAE recommended practice, J1213/1," tech. rep., Society of Automotive Engineers, September 1997.

[19] I. Broster, *Flexibility in dependable real-time communication.* PhD thesis, University of York, 2003.

[20] K. Tindell, A. Burns, and A. J. Wellings, "Calculating controller area network (CAN) message response times," *Control Engineering Practice*, vol. 3, no. 8, pp. 1163–1169, 1995.

[21] S. Punnekkat, H. Hansson, and C. Norstrom, "Response time analysis under errors for CAN," in *Real-Time Technology and Applications Symposium, 2000. RTAS 2000. Proceedings. Sixth IEEE*, pp. 258–265, IEEE, 2000.

[22] L. M. P. de Almeida, *Flexibility and timeliness in fieldbus-based real-time systems.* 1999.

[23] J. P. Thomesse, "A review of the fieldbuses," *Annual reviews in Control*, vol. 22, pp. 35–45, 1998.

[24] A. Burns and A. J. Wellings, *Real-time systems and programming languages.* Addison-Wesley, 1998.

[25] N. A. A. B. M. Richardson and A. Wellings, "Hard real-time scheduling: The deadline-monotonic approach1," in *Proceedings of the 8th IEEE Workshop on Real-time Operating Systems and Software*, Citeseer, 1991.

[26] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM (JACM)*, vol. 20, no. 1, pp. 46–61, 1973.

[27] G. K. Manacher, "Production and stabilization of real-time task schedules," *Journal of the ACM (JACM)*, vol. 14, no. 3, pp. 439–465, 1967.

[28] P. Richards, "Timing properties of multiprocessor systems. tech nical paper rep. no," tech. rep., TD-B60-27. Tech. Operations, Inc., Burlington, Massachusetts, 1960.

[29] B. Hedenetz and R. Belschner, "Brake-by-wire without mechanical backup by using a TTP-communication network," tech. rep., SAE Technical Paper, 1998.

[30] S. Amberkar, F. Bolourchi, J. Demerly, and S. Millsap, "A control system methodology for steer by wire systems," tech. rep., SAE Technical Paper, 2004.

[31] A. Avizienis, J.-C. Laprie, and B. Randell, "Fundamental concepts of dependability," *Department of Computing Science Technical Report Series*, 2001.

[32] Wikipedia, "Dependability — Wikipedia, the free encyclopedia," 2022. [Online; accessed 02-January-2022].

[33] H. Kopetz and G. Grunsteidl, "TTP – a protocol for fault-tolerant real-time systems," *Computer*, vol. 27, no. 1, pp. 14–23, 1994.

[34] H. Kopetz, "Should responsive systems be event-triggered or time-triggered?," *IEICE Transactions on Information and Systems*, vol. 76, pp. 1325–1332, 1993.

[35] R. Obermaisser, *Event-triggered and time-triggered control paradigms*, vol. 22. Springer Science & Business Media, 2004.

[36] L. Lamport, "Using time instead of timeout for fault-tolerant distributed systems.," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 6(2), pp. 254–280, 1984.

[37] P. Bosch, "50, d-700 stuttgart 1," *CAN Specification version 2.0 edition*, 1991.

[38] I. Standard, "11898: Road vehicles interchange of digital information controller area network (CAN) for high-speed communication," *International Standards Organization, Switzerland*, 1993.

[39] "Devicenet specifications, 2nd. ed. boca raton, 1997."

[40] H. Zeltwanger, *CANopen.* VDE-Verlag, 2001.

[41] I. ISO, "Iec 7498-1 information technology-open systems interconnection-basic reference model: The basic model," *International, Organization for Standardization*, vol. 2, 1994.

[42] P. Boait, *Open Systems Interconnection.* Macmillan International Higher Education, 1988.

[43] C. Specification, "Version 2.0," *Robert Bosch GmbH*, vol. 27, 1991.

[44] I. Standard, "11898: Road vehicles— interchange of digital information—controller area network (can) for high-speed communication," *International Standards Organization, Switzerland*, 1993.

[45] T. Schumann, "Canopen in industrial vehicles," tech. rep., SAE Technical Paper, 2002.

[46] L.-B. Fredriksson, "A can kingdom," *Mölndal, Sweden: Kvaser AB*, 1995.

[47] N. Navet, "Controller area network [automotive applications]," *IEEE Potentials*, vol. 17, no. 4, pp. 12–14, 1998.

[48] A. E. E. Committee *et al.*, "ARINC 629: IMA multi-transmitter databus parts 1-4," *Aeronautical radio, Inc., Annapolis, Maryland.–Octobre*, 1990.

[49] A. Specification, "651: Design guidance for integrated modular avionics, ser," *ARINC report. Airlines Electronic Engineering Committee (AEEC) and Aeronautical Radio Inc*, 1991.

[50] S. Berger, "Arinc 629 digital communication system application on the 777 and beyond," *Microprocessors and Microsystems*, vol. 20, no. 8, pp. 463–471, 1997.

[51] J. Azevedo and N. Cravoisy, "The worldfip protocol," *WorldFIP Organisation*, 1996.

[52] P. Nutzerorganisation eV, "Profibus technology and application-system description," 2002.

[53] AEEC, “Design guidance for integrated modular avionics,” *ARINC 651 Report*, 1997.

[54] J. Azevedo and N. Cravoisy, “The worldfip protocol,” *J. De Azevedo (Version1), N. Cravoisy (Version 2)*, vol. 2, 1998.

[55] N. C. Audsley and A. Grigg, “Timing analysis of the arinc 629 databus for real-time applications,” *Microprocessors and Microsystems*, vol. 21, no. 1, pp. 55–61, 1997.

[56] H. Koptez, “Real-time systems: design principles for distributed embedded applications,” 1997.

[57] H. Kopetz and K. Kim, “Temporal uncertainties in interactions among real-time objects,” in *Proceedings Ninth Symposium on Reliable Distributed Systems*, pp. 165–174, IEEE, 1990.

[58] K. Kim and H. Kopetz, “A real-time object model rto. k and an experimental investigation of its potentials,” in *Proceedings Eighteenth Annual International Computer Software and Applications Conference (COMPSAC 94)*, pp. 392–402, IEEE, 1994.

[59] H. Kopetz and W. Ochsenreiter, “Clock synchronization in distributed real-time systems,” *Computers, IEEE Transactions on*, vol. 100, no. 8, pp. 933–940, 1987.

[60] J. Rushby, “An overview of formal verification for the time-triggered architecture,” in *International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, pp. 83–105, Springer, 2002.

[61] H. Kopetz and R. Nossal, “The cluster compiler a tool for the design of time-triggered real-time systems,” in *Proceedings of the ACM SIGPLAN 1995 workshop on Languages, compilers, & tools for real-time systems*, pp. 108–116, 1995.

[62] H. Kopetz and G. Bauer, “The time-triggered architecture,” *Proceedings of the IEEE*, vol. 91, no. 1, pp. 112–126, 2003.

[63] H. Kopetz, F. Lohnert, W. Merker, and G. Pauthner, “An architecture for a maintainable real time system (MARS),” tech. rep., Technical Report TU Berlin, 1982.

[64] H. Curtis and R. France, “Time triggered protocol (TTP/C): A safety-critical system protocol,” *EE382C Literature Survey*, vol. 24, 1999.

[65] SAE, “Class c application requirement considerations, SAE recommended practice, J2056/1,” tech. rep., Society of Automotive Engineers, June 1993.

[66] F. Seidel, “X-by-wire,” in *Operation Systems, Chemnitz University of Technology, In seminar Transportation Systems*, 2009.

[67] S. Poledna, W. Ettlmayr, and M. Novak, “Communication bus for automotive applications,” in *Solid-State Circuits Conference, 2001. ESSCIRC 2001. Proceedings of the 27th European*, pp. 482–485, IEEE, IEEE.

[68] H. Kopetz and G. Grunsteidl, “Ttp-a time-triggered protocol for fault-tolerant real-time systems,” in *FTCS-23 The twenty-third international symposium on fault-tolerant computing*, pp. 524–533, IEEE, 1993.

[69] Kopetz, H. and Hexel, R. and Krüger, A. and Millinger, D. and Nossal, R. and Pallierer, R. and Steininger, A. and Temple, C. and Führer, T. and Krug, M., “A prototype implementation of a TTP/C controller,” tech. rep., SAE Technical Paper, 1997.

[70] B. Andersson, E. Tovar, and N. Pereira, “Analysing tdma with slot skipping,” in *Real-Time Systems Symposium, 2005. RTSS 2005. 26th IEEE International*, 2005.

[71] C. Li, M. Nicholas, and Q. Zhou, “A new real-time network protocol - node order protocol,” in *Proceedings of 11th Real Time Linux Workshop*, 2009.

[72] Berwanger, “et al. FlexRay the communication system for advanced automotive control systems,” *SAE Transactions*, vol. Vol. 110(7), pp. SAE Press, pp. 303–314, 2001.

[73] J. Berwanger, M. Peller, and R. Grießbach, "Byteflight a new protocol for safety critical applications," in *Proceedings of the 28th FISITA World Automotive Congress. Seoul, Korea: FISITA*, 2000.

[74] H. Kopetz, "A comparison of TTP/C and FlexRay," *Inst. for Computer Eng., Vienna*, 2001.

[75] T. Pop, P. Pop, P. Eles, Z. Peng, and A. Andrei, "Timing analysis of the FlexRay communication protocol," *Real-time systems*, vol. 39, no. 1-3, pp. 205–235, 2008.

[76] Y. Wang, Y. Xu, and Y. Xu, "Efficient utilization of flexray network using parameter optimization method," *International Journal of Engineering and Technology*, vol. 8, no. 6, 2016.

[77] J. Dvořák and Z. Hanzálek, "Using two independent channels with gateway for flexray static segment scheduling," *IEEE Transactions on Industrial Informatics*, vol. 12, no. 5, pp. 1887–1895, 2016.

[78] T.-Y. Lee, I.-A. Lin, J.-J. Wang, and J.-T. Tsai, "A reliability scheduling algorithm for the static segment of flexray on vehicle networks," *Sensors*, vol. 18, no. 11, p. 3783, 2018.

[79] N. Kumar and A. Mondal, "Timing analysis of precedence constraint messages scheduled with slot multiplexing over dynamic segment of flexray," *IEEE Transactions on Automation Science and Engineering*, vol. 17, no. 1, pp. 222–236, 2019.

[80] I. ISO, "11898-4-road vehicles-controller area network (can)-part 4: Time-triggered communication," *International Standard Organization*, pp. 11898–4, 2000.

[81] S. AS6802, "Time-triggered ethernet," *SAE International*, 2011.

[82] H. Yin, H. Jia, H. Qi, X. Ji, X. Xie, and W. Gao, "A hardware-efficient multi-resolution block matching algorithm and its vlsi architecture for high

definition mpeg-like video encoders," *IEEE TRANSACTIONS on circuits and systems for video technology*, vol. 20, no. 9, pp. 1242–1254, 2010.

[83] *"Institute of Electrical and Electronics Engineers, Audio/Video Bridging", in The Audio/Video Bridging Task Group, IEEE, 2011. [Online]. Available: http : //www.ieee802.org/1/pages/tsn.html.*

[84] ""audio video bridging (avb)", arista networks, tech. rep., 2009.," tech. rep.

[85] I. . W. Group *et al.*, "Ieee standard for local and metropolitan area networks timing and synchronisation for time sensitive applications in bridged local area networks," tech. rep., IEEE Std 802.1AS, 2011.

[86] I. . W. Group *et al.*, "Ieee standard for local and metropolitan area networks virtual bridged local area networks amendment 14: Stream reservation protocol (srp)," tech. rep., IEEE Std 802.1Qat Pages (1-119), 2010.

[87] I. . W. Group *et al.*, "Ieee standard for local and metropolitan area networks virtual bridged local area networks amendment 12: Forwarding and queuing enhancement for time sensitive streams," tech. rep., IEEE Std 802.1Qav, 2010.

[88] I. . W. Group *et al.*, "Ieee standard for local and metropolitan area networks audio video bridging (avb) systems," tech. rep., IEEE Std 802.1BA Pages (1-45), 2011.

[89] *"Institute of Electrical and Electronics Engineers, IEEE 1588 Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems", IEEE, 2008. [Online]. Available: https : / / standards . ieee.org/findstds/interps/1588-2008.html.*

[90] L. Zhao, F. He, and J. Lu, "Comparison of afdx and audio video bridging forwarding methods using network calculus approach," in *2017 IEEE/AIAA 36th Digital Avionics Systems Conference (DASC)*, pp. 1–7, IEEE, 2017.

[91] E. Heidinger, F. Geyer, S. Schneele, and M. Paulitsch, “A performance study of audio video bridging in aeronautic ethernet networks,” in *7th IEEE International Symposium on Industrial Embedded Systems (SIES’12)*, pp. 67–75, IEEE, 2012.

[92] *”Institute of Electrical and Electrnoics Engineers, Time Sensitive Networking”, in Time-Sensitive Networking Task Group, IEEE 2017. [Online]. Avaibale: http://www.ieee802.org/1/pages/tsn.html.*

[93] P. Meyer, T. Steinbach, F. Korf, and T. C. Schmidt, “Extending ieee 802.1 avb with time-triggered scheduling: A simulation study of the coexistence of synchronous and asynchronous traffic,” in *2013 IEEE Vehicular Networking Conference*, pp. 47–54, IEEE, 2013.

[94] *”Institute of Electrical and Electrnoics Engineers, Time Sensitive Networking, Inc. 802.1Qbv - Enhancement for Scheduled Traffic”, in Time-Sensitive Networking Task Group., IEEE, 2016. [Online]. Avaibale: http://www.ieee802.org/1/pages/802.1bv.html.*

[95] S. S. Craciunas, R. S. Oliver, M. Chmelík, and W. Steiner, “Scheduling real-time communication in ieee 802.1 qbv time sensitive networks,” in *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, pp. 183–192, 2016.

[96] *”Institute of Electrical and Electrnoics Engineers, Time Sensitive Networking, Inc. 802.1AS-Rev - Timing and Synchronisation for Time-Sensitive Applications”, in Time-Sensitive Networking Task Group., IEEE, 2017. [Online]. Avaibale: http://www.ieee802.org/1/pages/802.1AS-rev.html.*

[97] M. Pahlevan, “Time sensitive networking for virtualized integrated real-time systems,” 2020.

[98] A. Nasrallah, A. S. Thyagaturu, Z. Alharbi, C. Wang, X. Shao, M. Reisslein, and H. Elbakoury, “Performance comparison of ieee 802.1 tsn time aware shaper (tas) and asynchronous traffic shaper (ats),” *IEEE Access*, vol. 7, pp. 44165–44181, 2019.

[99] L. Zhao, P. Pop, and S. S. Craciunas, "Worst-case latency analysis for ieee 802.1 qbv time sensitive networks using network calculus," *Ieee Access*, vol. 6, pp. 41803–41815, 2018.

[100] *"Institute of Electrical and Electrnoics Engineers, Inc. 802.1Qci - Per stream Filtering and Policing", in Time-Sensitive Networking Task Group., IEEE, 2016. [Online]. Avaibale: http://www.ieee802.org/1/pages/802.1ci.html.*

[101] *"Institute of Electrical and Electrnoics Engineers, Inc. 802.1CB - Frame Replication and Elimination for Reliability", in Time-Sensitive Networking Task Group., IEEE, 2017. [Online]. Avaibale: http://www.ieee802.org/1/files/private/cb-drafts/d2/802-1CB-D2-9.pdf.*

[102] I. . W. Group *et al.*, "Ieee standard for local and metropolitan area network bridges and bridged networks," tech. rep., Technical Report Std 802.1 Q-2018. IEEE. 1–1993 pages.(Revision of IEEE Std), 2018.

[103] I. . W. Group *et al.*, "Ieee standard for local and metropolitan area networks frame replication and elimination for reliability," tech. rep., Technical Report IEEE Std 802.1CB-2017 pages.(1-102), 2017.

[104] *"Institute of Electrical and Electrnoics Engineers, Inc. 802.1Qca - Path Control and Reservation", in Time Sensitive Networking Task Group, IEEE, 2015.*

[105] P. Pop, M. L. Raagaard, S. S. Craciunas, and W. Steiner, "Design optimisation of cyber-physical distributed systems using ieee time-sensitive networks," *IET Cyber-Physical Systems: Theory & Applications*, vol. 1, no. 1, pp. 86–94, 2016.

[106] L. Bingqian and W. Yong, "Hybrid-ga based static schedule generation for time-triggered ethernet," in *2016 8th IEEE International Conference on Communication Software and Networks (ICCSN)*, pp. 423–427, IEEE, 2016.

[107] A. M. Kentis, M. S. Berger, and J. Soler, "Effects of port congestion in the gate control list scheduling of time sensitive networks," in *2017 8th*

International Conference on the Network of the Future (NOF), pp. 138–140, IEEE, 2017.

[108] V. Gavriluţ, L. Zhao, M. L. Raagaard, and P. Pop, “Avb-aware routing and scheduling of time-triggered traffic for tsn,” *Ieee Access*, vol. 6, pp. 75229–75243, 2018.

[109] F. Smirnov, M. Glaß, F. Reimann, and J. Teich, “Optimizing message routing and scheduling in automotive mixed-criticality time-triggered networks,” in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2017.

[110] E. Schweissguth, P. Danielis, D. Timmermann, H. Parzyjegla, and G. Mühl, “Ilp-based joint routing and scheduling for time-triggered networks,” in *Proceedings of the 25th International Conference on Real-Time Networks and Systems*, pp. 8–17, 2017.

[111] J. Falk, F. Dürr, and K. Rothermel, “Exploring practical limitations of joint routing and scheduling for tsn with ilp,” in *2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pp. 136–146, IEEE, 2018.

[112] N. G. Nayak, F. Dürr, and K. Rothermel, “Time-sensitive software-defined network (tssdn) for real-time applications,” in *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, pp. 193–202, 2016.

[113] E. Kyriakakis, J. Sparsø, and M. Schoeberl, “Implementing time-triggered communication over a standard ethernet switch,” in *Proceedings of the Workshop on Fog Computing and the IoT*, pp. 21–25, 2019.

[114] R. Ghosh, R. Pragathi, S. Ullas, and S. Borra, “Intelligent transportation systems: A survey,” in *2017 International Conference on Circuits, Controls, and Communications (CCUBE)*, pp. 160–165, IEEE, 2017.

[115] A. Luckow and K. Kennedy, “Data infrastructure for intelligent transportation systems,” in *Data Analytics for Intelligent Transportation Systems*, pp. 113–129, Elsevier, 2017.

[116] M. Alam, J. Ferreira, and J. Fonseca, “Introduction to intelligent transportation systems,” in *Intelligent transportation systems*, pp. 1–17, Springer, 2016.

[117] F. Arena and G. Pau, “An overview of vehicular communications,” *Future Internet*, vol. 11, no. 2, p. 27, 2019.

[118] K. Kiela, V. Barzdenas, M. Jurgo, V. Macaitis, J. Rafanavicius, A. Vasjanov, L. Kladovscikov, and R. Navickas, “Review of v2x–iot standards and frameworks for its applications,” *Applied Sciences*, vol. 10, no. 12, p. 4314, 2020.

[119] Q. Pei, B. Kang, L. Zhang, K.-K. R. Choo, Y. Zhang, and Y. Sun, “Secure and privacy-preserving 3d vehicle positioning schemes for vehicular ad hoc network,” *EURASIP Journal on Wireless Communications and Networking*, vol. 2018, no. 1, pp. 1–12, 2018.

[120] Z. El-Rewini, K. Sadatsharan, D. F. Selvaraj, S. J. Plathottam, and P. Ranganathan, “Cybersecurity challenges in vehicular communications,” *Vehicular Communications*, vol. 23, p. 100214, 2020.

[121] “Standard for information technology - telecommunications and information exchange between systems - local and metropolitan area networks - specific requirements - part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications..”

[122] “802.11p part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications: Amendment 7: Wireless access in vehicular environment..”

[123] K. Bilstrup, E. Uhlemann, E. Ström, and U. Bilstrup, “On the ability of the 802.11 p mac method and stdma to support real-time vehicle-to-vehicle

communication," *EURASIP Journal on Wireless Communications and Networking*, vol. 2009, pp. 1–13, 2009.

[124] K. Sjoberg, E. Uhlemann, and E. G. Strom, "How severe is the hidden terminal problem in vanets when using csma and stdma?," in *2011 IEEE vehicular technology conference (VTC Fall)*, pp. 1–5, IEEE, 2011.

[125] N. Suri, C. J. Walter, and M. M. Hugue, *Advances in ultra-dependable distributed systems.* IEEE Computer Society Press, 1994.

[126] J. Lee and H. Shin, "A variable bandwidth allocation scheme for ethernet-based real-time communication," in *Proc. of the First International Workshop on Real-Time Computing Systems and Applications*, pp. 28–32, 1994.

[127] F. Heilmann and G. Fohler, "Impact of time-triggered transmission window placement on rate-constrained traffic in ttethernet networks," *ACM SIGBED Review*, vol. 15, no. 3, pp. 7–12, 2018.

[128] V. Eramo, F. G. Lavacca, F. Valente, A. Pisculli, and S. Caporossi, "Simulation and experimental evaluation of a flexible time triggered ethernet architecture applied in satellite nano/micro launchers," *Aerospace*, vol. 5, no. 3, p. 84, 2018.

[129] M. Sugihara, "Dynamic slot multiplexing under operating modes for tdma-based real-time networking systems," *Electronics*, vol. 9, no. 2, p. 224, 2020.

[130] R. M. Vaz, K. N. Hodel, M. M. Santos, B. A. Arruda, M. L. Netto, and J. F. Justo, "Vehicular communications," 2020.

[131] P.-S. Murvay and B. Groza, "Efficient physical layer key agreement for flexray networks," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 9, pp. 9767–9780, 2020.

[132] J. Falk, F. Dürr, and K. Rothermel, "Time-triggered traffic planning for data networks with conflict graphs.," in *RTAS*, pp. 124–136, 2020.

[133] K. Krüger, N. Vreman, R. Pates, M. Maggio, M. Volp, and G. Fohler, "Randomization as mitigation of directed timing inference based attacks on time-triggered real-time systems with task replication," *LITES: Leibnitz Transactions on Embedded Systems*, 2021.

[134] E. Kyriakakis, J. Sparsø, P. Puschner, and M. Schoeberl, "Synchronizing real-time tasks in time-triggered networks," in *2021 IEEE 24th International Symposium on Real-Time Distributed Computing (ISORC)*, pp. 11–19, IEEE, 2021.

[135] N. Ragesh, "Data traffic in new generation vehicles," *CSI Communications*, vol. 35, p. 12, 2012.

[136] O. El Marai and T. Taleb, "Smooth and low latency video streaming for autonomous cars during handover," *IEEE Network*, vol. 34, no. 6, pp. 302–309, 2020.

[137] N.-S. N. Ismail, F. Yunus, S. H. Ariffin, A. Shahidan, R. A. Rashid, W. Embong, N. Fisal, and S. Yusof, "Mpeg-4 video transmission using distributed tdma mac protocol over ieee 802.15. 4 wireless technology," in *2011 Fourth International Conference on Modeling, Simulation and Applied Optimization*, pp. 1–6, IEEE, 2011.

[138] L. R. Pinto, L. Almeida, H. Alizadeh, and A. Rowe, "Aerial video stream over multi-hop using adaptive tdma slots," in *2017 IEEE Real-Time Systems Symposium (RTSS)*, pp. 157–166, IEEE, 2017.

[139] D. Saxena and V. Raychoudhury, "Design and verification of an ndn-based safety-critical application: A case study with smart healthcare," *ieee transactions on systems, man, and cybernetics: systems*, vol. 49, no. 5, pp. 991–1005, 2019.

[140] D. Chen, R. Hexel, and F. R. Raja, "Incus: A communication protocol for safety critical distributed real time systems," in *The 20th Asia-Pacific Conference on Communication (APCC2014)*, pp. 309–314, IEEE, 2014.

[141] G. Bauer and M. Paulitsch, "An investigation of membership and clique avoidance in ttp/c," in *Proceedings 19th IEEE Symposium on Reliable Distributed Systems SRDS-2000*, pp. 118–124, IEEE, 2000.

[142] H. Pfeifer, D. Schwier, and F. W. von Henke, "Formal verification for time-triggered clock synchronization," in *Dependable Computing for Critical Applications 7*, pp. 207–226, Jan 1999.

[143] T. Hase, W. Hintermaier, A. Frey, T. Strobel, U. Baumgarten, and E. Steinbach, "Influence of image/video compression on night vision based pedestrian detection in an automotive application," in *Vehicular Technology Conference (VTC Spring), 2011 IEEE 73rd*, pp. 1–5, IEEE, 2011.

[144] W. Kubinger, H. Hemetsberger, and J. Kogler, "Platooning platform for analyzing embedded control algorithms," *Annals of DAAAM & Proceedings*, pp. 211–213, 2005.

[145] J.-E. Kallhammer, D. Eniksson, G. Granlund, M. Felsberg, A. Moe, B. Johansson, J. Wiklund, and P.-E. Forssén, "Near zone pedestrian detection using a low-resolution fir sensor," in *Intelligent Vehicles Symposium, 2007 IEEE*, pp. 339–345, IEEE, 2007.

[146] D. Chen, R. Hexel, and F. R. Raja, "Engineering real-time communication through time-triggered subsumption," in *Proceedings of the 11th International Conference on Evaluation of Novel Software Approaches to Software Engineering*, pp. 272–281, SCITEPRESS-Science and Technology Publications, Lda, 2016.

[147] H. Kopetz, R. Nossal, R. Hexel, A. Krüger, D. Millinger, R. Pallierer, C. Temple, and M. Krug, "Mode handling in the time-triggered architecture," *Control Engineering Practice*, vol. 6, no. 1, pp. 61–66, 1998.

[148] M. Ahuja, A. D. Kshemkalyani, and T. Carlson, "A basic unit of computation in distributed systems.," in *ICDCS*, pp. 12–19, 1990.

[149] V. Estivill-Castro and R. Hexel, "Module isolation for efficient model checking and its application to FMEA in model-driven engineering," in *ENASE 8th International Conference on Evaluation of Novel Approaches to Software Engineering*, (Angers Loire Valley, France), pp. 218–225, INSTCC, July 4th-6th 2013.

[150] D. C. Schmidt, "Guest editor's introduction: Model-driven engineering," *IEEE Computer*, vol. 39, no. 2, pp. 25–31, 2006.

[151] A. Bucchiarone, J. Cabot, R. F. Paige, and A. Pierantonio, "Grand challenges in model-driven engineering: an analysis of the state of the research," *Software and Systems Modeling*, vol. 19, no. 1, pp. 5–13, 2020.

[152] A. Idani, Y. Ledru, and G. Vega, "Alliance of model-driven engineering with a proof-based formal approach," *Innovations in Systems and Software Engineering*, vol. 16, pp. 289–307, 2020.

[153] C. D. N. Damasceno and D. Strüber, "Quality guidelines for research artifacts in model-driven engineering," in *2021 ACM/IEEE 24th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pp. 285–296, IEEE, 2021.

[154] V. Estivill-Castro and R. Hexel, "Correctness by construction with logic-labeled finite-state machines – comparison with Event-B," in *Proc. 23rd Australian Software Engineering Conference (ASWEC)*, pp. 38–47, IEEE, 2014.

[155] D. Billington, V. Estivill-Castro, R. Hexel, and A. Rock, "Requirements engineering via non-monotonic logics and state diagrams," in *Evaluation of Novel Approaches to Software Engineering (ENASE selected papers)*, vol. 230 of *Communications in Computer and Information Science*, (Athens, Greece), pp. 121–135, Springer Verlag, 22-24 July 2011.

[156] V. Estivill-Castro, R. Hexel, and M. McColl, "High-level executable models of reactive real-time systems with logic-labelled finite-state machines and

fpgas," in *2018 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, pp. 1–8, IEEE, 2018.

[157] F. Grubb, V. Estivill-Castro, and R. Hexel, "Llfsms on the pru: Executable and verifiable software models on a real-time microcontroller," in *International Conference On Systems Engineering*, pp. 391–402, Springer, 2021.

[158] V. Estivill-Castro, R. Hexel, and D. A. Rosenblueth, "Efficient modelling of embedded software systems and their formal verification," in *The 19th Asia-Pacific Software Engineering Conference (APSEC 2012)* (K. R. Leung and P. Muenchaisri, eds.), (Hong Kong), pp. 428–433, IEEE Computer Society, Conference Publishing Services, December 2012.

[159] M. Lindgren, "Practical verification of stateful embedded c code using finite state machines and vcc," 2020.

[160] N. Krafczyk and J. Peleska, "Exhaustive property oriented model-based testing with symbolic finite state machines," in *International Conference on Software Engineering and Formal Methods*, pp. 84–102, Springer, 2021.

[161] R. Brooks *et al.*, "A robust layered control system for a mobile robot," *IEEE Journal of Robotics and Automation*, vol. 2, no. 1, pp. 14–23, 1986.

[162] L. P. Kaelbling, "An architecture for intelligent reactive systems," in *Morgan Kaufmann, Proceedings of the 1986 Workshop: Reasoning about Actions and Plans, Editors: Georgeff, M, Lansky, A*, vol. 30, pp. 395–410, 1987.

[163] D. W. Payton, "An architecture for reflexive autonomous vehicle control," in *Proc. IEEE International Conference on Robotics and Automation.*, vol. 3, pp. 1838–1845, IEEE, 1986.

[164] R. C. Arkin, "Motor schema based navigation for a mobile robot: An approach to programming by behavior," in *Robotics and Automation. Proceedings. 1987 IEEE International Conference on*, vol. 4, pp. 264–271, Mar 1987.

[165] J. Connell, “Creature design with the subsumption architecture.,” in *IJCAI*, vol. 87, pp. 1124–1126, 1987.

[166] R. A. Brooks, J. Connell, and P. Ning, “Herbert: A second generation mobile robot,” *MIT AI Memo 1016*, 1988.

[167] M. J. Mataric, “Qualitative sonar based environment learning for mobile robots,” in *Proc. Advances in Intelligent Robotics Systems Conference*, pp. 305–315, International Society for Optics and Photonics, 1990.

[168] R. A. Brooks, “Micro-brains for micro-brawn: Autonomous microbots,” in *IEEE Micro Robots and Teleoperators Workshop: An investigation of micromechanical structures, actuators and sensors, Hyannis, MA*, 1987.

[169] P. Ögren and C. I. Sprague, “Behavior trees in robot control systems,” *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 5, 2021.

[170] K. Othman, *Towards the vision of a social robot in every home: A navigation strategy via enhanced subsumption architecture.* PhD thesis, Applied Sciences: School of Mechatronic Systems Engineering, 2020.

[171] Z. Chu, “Development of hybrid control architecture for a small autonomous underwater vehicle,” in *Fundamental Design and Automation Technologies in Offshore Robotics*, pp. 161–175, Elsevier, 2020.

[172] D. Billington, V. Estivill-Castro, R. Hexel, and A. Rock, “Requirements engineering via non-monotonic logics and state diagrams,” in *Evaluation of Novel Approaches to Software Engineering*, pp. 121–135, Springer, 2011.

[173] V. Estivill-Castro and R. Hexel, “Arrangements of finite-state machines semantics, simulation, and model checking,” in *International Conference on Model-Driven Engineering and Software Development MODELSWARD* (S. Hammoudi, L. Ferreira Pires, J. Filipe, and R. César das Neves, eds.), (Barcelona, Spain), pp. 182–189, SCITEPRESS Science and Technology Publications, 19-21 February 2013.

[174] I. Jacobson and E. Seidewitz, “A new software engineering: What happened to the promise of rigorous, disciplined, professional practices for software development?,” *ACM-Queue*, vol. 12, October 2014.

[175] J. Erickson and K. Siau, “Can UML be simplified? practitioner use of UML in separate domains,” in *proceedings EMMSAD*, vol. 7, pp. 87–96, 2007.

[176] G. Reggio, M. Leotta, F. Ricca, and D. Clerissi, “What are the used UML diagrams? A preliminary survey,” in *Proceedings of the 3rd International Workshop on Experiences and Empirical Studies in Software Modeling co-located with 16th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2013)* (M. R. V. Chaudron, M. Genero, S. Abrahão, and L. Pareto, eds.), vol. 1078 of *CEUR Workshop Proceedings*, pp. 3–12, October 1st 2013.

[177] H. Kopetz and G. Bauer, “The time-triggered architecture,” *Proceedings of the IEEE*, vol. 91, no. 1, pp. 112–126, 2003.

[178] D. Chen, R. Hexel, and F. R. Raja, “INCUS: A communication protocol for safety-critical distributed real-time systems,” in *proceedings of 20th Asia-Pacific Conference on Communications (APCC), Pattaya, Thailand*, October 2014.

[179] V. Estivill-Castro, R. Hexel, and C. Lusty, “High performance relaying of C++11 objects across processes and logic-labeled finite-state machines,” in *Simulation, Modeling, and Programming for Autonomous Robots - 4th International Conference, SIMPAR 2014* (D. Brugali, J. F. Broenink, T. Kroeger, and B. A. MacDonald, eds.), vol. 8810 of *Lecture Notes in Computer Science*, (Bergamo, Italy), pp. 182–194, Springer, October 20th-23rd 2014.

[180] H. Kopetz, *Real-Time Systems*. Kluwer Academic Publishers, 1997.

[181] C. Temple, “Avoiding the babbling-idiot failure in a time-triggered communication system,” in *Digest of Papers. Twenty-Eighth Annual International Symposium on Fault-Tolerant Computing (Cat. No. 98CB36224)*, pp. 218–227, IEEE, 1998.

[182] O. Daniel and O. Roman, "Fault injection framework for assessing fault containment of ttethernet against babbling idiot failures," in *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*, pp. 1–6, IEEE, 2018.

[183] D. Powell *et al.*, *Delta-4, A generic architecture for dependable distributed computing*, vol. 199. Springer, 1991.

[184] H. Kopetz, "Fault containment and error detection in the time-triggered architecture," in *The Sixth International Symposium on Autonomous Decentralized Systems, 2003. ISADS 2003.*, pp. 139–146, IEEE, 2003.

[185] I. Broster and A. Burns, "An analysable bus-guardian for event-triggered communication," in *RTSS 2003. 24th IEEE Real-Time Systems Symposium, 2003*, pp. 410–419, IEEE, 2003.

[186] J. Almeida, J. Ferreira, and A. S. Oliveira, "A medium guardian for enhanced dependability in safety-critical wireless systems," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 3, pp. 965–976, 2018.

[187] C. Specification, "Bosch," *Robert Bosch GmbH, Postfach*, vol. 50, 1991.

[188] A. L. Hopkins, T. B. Smith, and J. H. Lala, "Ftmp, a highly reliable fault-tolerant multiprocess for aircraft," *Proceedings of the IEEE*, vol. 66, no. 10, pp. 1221–1239, 1978.