Flexibility in Dependable Real-time Communication

Ian Broster

This thesis is submitted in partial fulfilment
of the requirements for the degree of
Doctor of Philosophy.

University of York

Department of Computer Science

August 2003

Abstract

THE ROLE OF THE COMMUNICATIONS BUS is fundamental in distributed real-time control systems. Such systems are increasingly used for critical functions in avionics, automotive and factory control situations, placing increased dependability and real-time constraints on the bus. Environmental influences such as electromagnetic interference are hard to avoid so a "flexible" bus may be able to provide active *fault tolerance*. However its effects on reliability and timeliness are difficult to predict.

This thesis contends that guaranteeing to meet *all* deadlines in communication is not only impractical, but often impossible, due to the unpredictability of environmental interference, no matter which type of electrical bus is used. However, many applications are capable of safe operation if a small number of communication deadlines are missed. In such systems, an analysable and reliable system can be achieved through the use of flexible fault tolerance.

Using CAN (a widely used bus protocol) as a basis, this thesis first shows how weakly-hard analysis (which considers timing behaviour over a number of invocations) can be applied to flexible bus scheduling. This allows consideration of more than just the worst case scenario, leading to analysable and predictable behaviour under severe environmental conditions. A second form of analysis based on a probabilistic fault model is used to provide accurate probabilities of failure, providing the facility to explore system behaviour analytically for fault scenarios which exceed normal behaviour. Finally, a simple extension to the CAN protocol, TCAN (Timely-CAN), is proposed which enforces timely recovery from faults by only using CAN message retransmission where it is useful to do so without imposing further delays on the bus. Hence the flexibility of CAN is exploited to provide fault tolerance, and both timeliness and predictability are achieved.

Contents

Abstract **3**

Table of Abbreviations **11**

Table of Symbols **13**

Acknowledgements **15**

Declaration **17**

1 Flexible and Dependable Communication **19**
1.1 Distribution 20
1.2 Environment 22
1.3 Flexibility 23
1.4 Real-time and Control Systems 24
1.5 Contribution 25

2 Real-time Communication **27**
2.1 Real-time Systems 27
2.2 Real-time Communication 32
2.3 Dependability 34
2.4 Flexibility in Real-time Communication 37
2.5 Controller Area Network 45
2.6 CAN Worst Case Response Time Analysis 58
2.7 Bus Faults on CAN 66
2.8 Scheduling on CAN 71
2.9 Summary 73

3 Weakly-hard Analysis of CAN **75**
3.1 Motivation 75
3.2 Intuition 77
3.3 Weakly-hard Theory 78
3.4 Guaranteed Weakly-hard Analysis of CAN 80
3.5 Incorporating Faults: Calculating $E_{i,k}(t)$ 83
3.6 Conversion to Weakly Hard Constraints 85

3.7 Critical Instants and Harmonic Periods . . . 85
3.8 Relaxing Clock Synchronisation . . . 88
3.9 Evaluating Weakly Hard . . . 89
3.10 Simulation of CAN . . . 101
3.11 Resilience of the Analysis . . . 105
3.12 Pessimism in Weakly-hard Analysis . . . 108
3.13 Summary . . . 109

4 Probabilistic Analysis of CAN 111
4.1 Worst Case Analysis and Fault Models . . . 111
4.2 Probabilistic Analysis Approaches . . . 114
4.3 A New Probabilistic Analysis of CAN . . . 123
4.4 Interpretation of the Distribution . . . 129
4.5 Probabilities, Complexity and ρ . . . 130
4.6 Implementation of the Probabilistic Algorithm . . . 133
4.7 Improving the Analysis . . . 134
4.8 Multiple Invocation Analysis . . . 139
4.9 Evaluating the Analysis . . . 140
4.10 Summary . . . 150

5 Predictable Failure 153
5.1 Approach and Justification . . . 153
5.2 Intuition . . . 155
5.3 The Timely-CAN Protocol . . . 156
5.4 Properties . . . 162
5.5 Strategies for Deadline Calculation . . . 164
5.6 Common Knowledge of Delivery Threshold Times . . . 170
5.7 Clock Synchronisation . . . 172
5.8 Implementation . . . 175
5.9 Evaluation . . . 177
5.10 Summary . . . 181

6 Conclusion 185
6.1 Missing Deadlines . . . 186
6.2 Exploiting Variation . . . 187
6.3 Exploiting Dynamic Fault-tolerance . . . 187
6.4 Exploiting Missed Deadlines . . . 188
6.5 Evaluation . . . 189
6.6 Exploitation of Results . . . 192
6.7 Future Work . . . 193
6.8 Concluding Remarks . . . 194

List of References 195

List of Tables

3.1 SAE Benchmark. 91
3.2 Non-harmonic Message Set. 94
3.3 Worst Case Response Times with Faults. 97
3.4 Percentage of Times that the Constraint $\left\langle {n \atop m} \right\rangle$ is Satisfied. 98
3.5 Analysis of deadlines missed in a row. 99
3.6 Percentage of Guaranteed Weakly Hard Constraints $\binom{n}{m}$ by Analysis. 99
3.7 Analysis of Deadlines Missed in a Row. 99
3.8 Comparison of Weakly-hard Analysis and Simulation at High Levels of Faults. 104
3.9 Comparison of Weakly-hard Constraints and Poisson Simulations. . . 107

4.1 Burns and Punnekkat Guarantees vs. Simulation Results. 118
4.2 Navet WCDFP Analysis. 123
4.3 Enumeration of Scenarios. 136
4.4 Peugeot Example Message Set. 141
4.5 Probabilistic Analysis results. 142
4.6 Probabilistic Analysis of One Frame. 147
4.7 Comparison of new Analysis, Previous Approach and Simulation. . . 148

5.1 TCAN and CAN Fault Injection Results. 178

List of Figures

2.1 Two Cycles of a TDMA schedule example. 38
2.2 CAN Protocol Layers. 48
2.3 CAN Data Frame Format. 52

3.1 Typical Distribution of Response Times. 78
3.2 Typical Distribution of Response Times with One Fault. 79
3.3 Virtual Task to Calculate the Idle Time. 83
3.4 Harmonic Periods. 86
3.5 Non-harmonic Periods. 86
3.6 Variation in Worst Case Response Times for One Fault in Hyperperiod 95
3.7 Cumulative Distribution of Response Times with One Fault. 96

4.1 Approximations to Burns' Approach: Comparison. 117
4.2 Modified Navet Analysis and Simulation Results. 124
4.3 Example Probability Tree for Simple Fault Model. 127
4.4 Interpretation of Results. 130
4.5 Analysis Algorithm. 133
4.6 Response Times for Improved Algorithm. 135
4.7 Probability Distribution of Response Times for All Frames. 143
4.8 Response Time Probability, Analysis vs. Simulation. 144
4.9 Probabilistic Analysis of all frames. 146
4.10 Analysis and Simulation for One Frame. 147
4.11 Comparison of Navet and New Probabilistic Analyses. 149

5.1 TCAN: Simple Example Showing How Losing Messages Can Improve Timeliness of Others. 158
5.2 Code for TCAN Transmitter. 160
5.3 Code for TCAN Receiver. 161
5.4 Possible Values for Deadlines. 167
5.5 Effect of Clock Skew. 173
5.6 Protected-TCAN. 174
5.7 Distribution of Lost Frames. 180
5.8 Distribution of Lost Frames for 3 Threshold Schemes. 181

Table of Abbreviations

Abbreviations used in this thesis, their meaning and index of the main pages where the abbreviations are used.

Abbreviation	Meaning	Page
CAN	Controller Area Network	24, 34
CAN2.0A	Standard Format CAN	51
CAN2.0B	Extended Format CAN	51
COTS	Commercial Off the Shelf	175
CPU	Central Processing Unit	75
CRC	Cyclic Redundancy Check	50, 56
CSMA	Carrier Sense Multiple Access	40
CSMA-CA	CSMA—Collision Avoidance	41
CSMA-CD	CSMA—Collision Detection	43
CSMA-DCR	CSMA—Deterministic Collision Resolution	44
DMPO	Deadline Monotonic Priority Ordering	71
EDF	Earliest Deadline First	71, 194
EMI	Electromagnetic Interference	23, 30
FTMP	Fault Tolerant Multiprocessor	22
FTT	Flexible Time Triggered	33, 42
FTTCAN	Flexible Time Triggered CAN	42, 72, 175
ISO	International Standards Organisation	47
LST-CAN	Latest Send Time CAN, now called TCAN	153
MAC	Media Access Control	37
OSI	Open System Interconnection	47

continued over...

...continued

Abbreviation	**Meaning**	**Page**
PID	Proportional, Integral, Derivative	60, 76
RMPO	Rate Monotonic Priority Order	71, 87, 141
SAE	Society of Automotive Engineers	90
TCAN	Timely Controller Area Network	26
TDMA	Time Division Multiple Access	33, 37, 190
TTCAN	Time Triggered CAN	45, 73, 153
TTP	Time Triggered Protocol	24, 34, 37
WCDFP	Worst Case Deadline Failure Probability	70, 119
WCRT	Worst Case Response Time	59, 62

Table of Symbols

Symbols used in this thesis, their meaning and index of the main pages where the symbols are used.

Symbol	Meaning	Page
A_i	Number of invocations of stream i in the hyperperiod	81
B_i	Worst case blocking	65, 82, 126
C_i	Longest frame length	61, 64
D_i	Deadline for a frame in stream i	61, 65, 94, 131, 165, 181
E	Maximum length of an error frame	67, 84, 125
$E_i(t)$	Worst case overhead due to faults in interval t	66, 67, 84, 125, 189
H	Length of the hyperperiod	81, 92
H_i	Length of the hyperperiod at level i	81, 93
$I_i(t)$	Worst case interference in interval t	65, 82, 89, 126
J_i	Worst case jitter	61, 66, 102, 115, 126
O_F	Offset for pseudo-periodic fault stream	84
O_i	Offset for frames in stream i	61, 170
R_i	Worst case response time of stream i	62, 64, 93, 115, 128, 155, 167, 180, 194

continued over...

...continued

Symbol	Meaning	Page
$R_{i,k}$	Worst case response time of the k^{th} invocation in stream i	81, 85
$R_{i\|K}$	Worst case response time for a frame in stream i if K faults delay the frame	120, 135, 137
S	Time of CAN inter-frame space	64, 82
$S_{i,k}$	Time of trigger event of k^{th} invocation in stream i	81, 84, 164
T_F	Minimum interarrival time between faults	67, 84, 92, 103
T_i	Period or minimum inter-arrival time	61, 66, 81, 85, 126, 167, 170
U	Total maximum utilisation	62, 81, 99
U_i	Maximum utilisation of stream i	62
X_i	TCAN delivery threshold for stream i	163, 166, 178, 194
$X_{i,k}$	TCAN delivery threshold for the k^{th} invocation in stream i	157, 164, 170, 174
$\delta_{i,k}$	Idle time before the k^{th} invocation in stream i	82
ρ	A very small, insignificant, probability used as a threshold	125, 127–133
ε	Bounded Clock Difference	67, 81, 125, 135
$\mathsf{hp}(i)$	Set of streams with higher priority to stream i	62, 65, 86
λ	Parameter of Poisson Distribution, mean number of faults per unit interval	103, 105
$\mathsf{lp}(i)$	Set of streams with lower priority to stream i	65
τ	Time of one bit on CAN bus	63, 65, 89
a_i	Number of invocations of stream i in the hyperperiod at level i	81, 87, 93

Acknowledgements

There are a number of people and organisations who have contributed directly and indirectly to this thesis. I wish to express my sincere gratitude to them all. In particular, my supervisor, Professor Alan Burns has provided invaluable guidance and advice throughout my years at university. Some of the research in this thesis has been performed jointly between myself and other people: Dr. Guillem Bernat of the University of York was involved with the chapter on weakly-hard analysis and collaborative work with Mr. Guillermo Rodrìguez-Navas from the Universitat de les Illes Balears formed the basis of the chapter on probabilistic analysis. I would also like to thank my colleagues in the Real-time Systems Group at the University of York for their support and informal discussions which form a vital part of the research environment. In particular, Mr. Michael Bennett, Dr. Iain Bate and Mr. George de Lima. I also wish to thank Dr. Julian Proenza of the Universitat de les Illes Balears for his encouragement and interest in this research work.

This work has been funded by Rolls-Royce plc and EPSRC through a CASE award. I would like to thank the people at Rolls-Royce who funded this work and especially Dr. Tim Kelly for arranging the partnership and also for his role as the internal assessor for this doctorate programme.

Finally, many thanks to Karen, my wife, for all her help, encouragement and support.

Declaration

Parts of this thesis have been published previously and in collaboration with other people. The work in Chapter 3 was done in collaboration with Dr. Guillem Bernat of the University of York and was published in 2002 at the Euromicro Real-time Systems Conference [31]. The work in Chapter 4 was done in collaboration with Mr. Guillermo Rodríguez-Navas of the Universitat de les Illes Balears and was published at in 2002 at the IEEE Real-time Systems Symposium [35]. The work in Chapter 5 was published previously at the 2001 Euromicro Real-time Systems Conference [33] and future work is to appear in the 2003 Emerging Technologies and Factory Automation Conference [57]. Related work has been also published at the 2001 Real-time Systems Symposium [32] and to appear at the 2003 Real-time Systems Symposium [34]. Except where otherwise stated, all material is the author's own.

1 Flexible and Dependable Communication

MODERN COMPUTER CONTROL SYSTEMS are more *distributed* today than ever before and the trend looks likely to continue. Crucial to a distributed system is the *communications* (sub)system—the network or bus system that links together the nodes in the system—its role is central in supporting the *functional*, *real-time* and *dependability* requirements of the system.

The functional requirements are difficult to characterise since they vary massively between different systems. However, one trend which may be observed is that modern systems have more and more demanding functional and non-functional requirements, involving increased bandwidth, dynamic loads, fault tolerance and reconfiguration [30, 19].

Many control systems perform functions with *real-time* requirements: the times at which they interact with their environment are important in some way. For example, an anti-lock braking system needs to respond to the wheels locking and the pressure on the brake pedal quickly enough to keep the vehicle under control.

The environment in which many distributed, embedded systems reside may be harsh, with extremes of electrical interference, vibration, temperature and pressure. An aircraft engine, for example, suffers severe vibration and very low temperatures and pressures in addition to electrical noise from many sources including [69] lightning, radar, radio, other aircraft and the engine itself.

In a distributed control system, the communications system is crucial, yet this is a component that is particularly at risk from environmental influences due to its long lengths of electrical cabling [121].

The rest of this chapter explores the key elements of distribution, the environment, flexibility and real-time control to provide motivation for the contributions in this thesis towards flexible, dependable communication.

1.1 Distribution

As stated earlier, a current trend in control systems is towards distribution, and in the context of control systems this leads to localised control. It is interesting to note the apparent change in direction, as the decreasing cost of computer hardware allows the advantages of computer control to be distributed around the system.

Traditionally (before any computer control), mechanical systems were entirely distributed: complex mechanical control was accomplished (with a great deal of innovation) using devices such as cams, levers and speed regulators. Amazing feats have been achieved using clockwork technology, the famous 'Mechanical Turk' mechanical chess player from the 18th Century illustrates the ingenious possibilities of mechanical control [151]. Through necessity, mechanical control must be performed where it is needed, distributed throughout the system.

Later, with the advent of electronics and computing, the advantages of software-based control became apparent. The complex (and often error-prone) mechanical devices were replaced by far simpler electronic sensors and actuators such as solenoids and servos, wired to a central computer system. This enabled complex yet reliable control to be implemented in software with relative ease. Many good examples of this architecture may be seen today, for example in factory automation, aircraft engines, household appliances and many commercial vehicles from around the 1980's. Of course, this architecture has many disadvantages too: not least the fact that there is one single, very complex point of failure. Modern computer systems are so complex, performing so many different functions in one place, that the resulting behaviour can become far from predictable.

The miniaturisation, availability and decreasing cost of computer based systems have been some of the most important drivers for the current trend back to distributed control. This time though, the localised control is computer-based rather then entirely

mechanical. This allows the advantages of computer-based control to be combined with localised control, providing a sound solution with the positive characteristics of both styles of architecture. There are other motives for distribution too [29], explained in the following.

Reduce wiring and wiring complexity: in a typical single processor scenario there may be many sensors spread around the system, each connected to the processing unit with its own point-to-point connection. Hence there is a lot of wiring, routed all over the system. This is complex to set up, heavy and expensive because it uses a lot of cable, and prone to faults which are then difficult to find and repair. If more than one processing unit is used so that the processing can be done near to the sensors and the processors connected together using a bus, then the length and weight of cable and the complexity of wiring are significantly reduced [89].

Reduce complexity in the nodes: as the number of nodes increases, the amount of functionality that each node must support can decrease. This results in both simpler hardware and simpler software. The positive effects of fewer 'devices' (per node), hardware interfaces and software components are numerous. Not least is the possibility to improve *predictability*, since smaller nodes allow more accurate and predictable control of resources.

Improve diagnostics: using multiple processors could provide the ability to pinpoint the source of faults [156]. By splitting one large and complex system into many smaller ones, an external monitoring system can monitor communications between them to diagnose a fault to a specific node which is cheap and simple to replace. This can be an improvement over diagnostic procedures on a single 'black box', where the box is very complex.

Provide more processing capacity: more processor bandwidth distributed across the system allows the possibility of more sophisticated control, distributed problem solving, advanced logging and diagnostic functions *etc.*

Allow composability and expansion: it is often easier to expand functionality by connecting a new node to a network than it is to modify existing nodes. Part of

the reason for this is that the interface is naturally well-defined, the possibilities for unseen dependencies such as shared memory violations, timing dependencies or competition for the CPU are reduced. However, composability is far from being a trivial problem [153].

Provide fault-tolerance: a distributed system has clear potential for implementing modular redundancy at the node level by replicating nodes. This has been done many times before in a variety of guises, see for example the replication in the FTMP architecture as far back as 1978 [65].

There are many examples of increasing distribution today, for example in a modern car, there may be approximately 100-150 small embedded computers connected together around the vehicle, performing functions such as engine management, assisted braking, climate control and entertainment. Forecasts suggest that this number is set to grow to over 1000 [101].

Brake-by-wire systems (with no hydraulic backup) are likely to be standard in future cars, "electromechanical brakes are far easier to build, are more serviceable, more reliable and the response time between the driver's foot and the brakes is faster" [60]. The existing research prototypes for in-vehicle automation possibilities make eye-opening reading [89]. However, with these advantages may come the responsibilities of safety-critical systems, where there is a need to provide reliable behaviour, even in the presence of a variety of faults [110]. Here lies the reason that these "Drive-by-Wire" systems are not yet in production vehicles; at least some part of this hesitation comes from uncertainty about the reliability of bus-based systems [89].

Other examples of increasing distribution include factory communication systems [162] and modular avionics [1, 77] where there is intense interest in dependable communication.

1.2 Environment

Dependable, distributed, embedded real-time systems have been designed and used successfully for many years. Such systems are seen in cars, factories, aircraft and

many other applications. A variety of different technologies have been used to implement these systems and there is little reason to doubt their dependability.

However, the environment in which these systems reside can be *unpredictable* and may have unpredictable effects on the system. The environment is often harsh with extremes of temperature, pressure and vibration. It may be electrically *noisy*, with high levels of electromagnetic radiation. EMI (electromagnetic interference) is generally considered to have been the "cause" of several UH-60 'Black Hawk' Army helicopter crashes [169] and it is suggested that EMI has been behind a number of other unexplained avionics disasters [49]. Many more examples of EMI related accidents can be found in a wide variety of domains, such as mining, off-shore oil platforms, sewerage processing, silicon chip manufacture, wheel-chair design and so on [69].

The communications sub-system in particular is vulnerable to electrical noise since it typically includes long lengths of cabling which are susceptible to perturbations due to EMI. It is very difficult to predict the interference that a system will be subjected to during its lifetime. Despite EMI modelling techniques, perhaps the only safe statement about predicting EMI is that it is unpredictable [69]. In an unpredictable environment, predictability and reliability are difficult to argue about. Can the reliability in an unpredictable environment be predicted?

1.3 Flexibility

Flexibility and *dependability* have often been regarded as opposing parameters in an engineering tradeoff [3, 155]; there is often not an obvious choice to be made. It has been argued that the best way to provide *dependability* is to make things more *static* [82], that is to remove any *dynamic behaviour* so that the exact sequence of events that will occur at run-time is determined in advance. Certainly, this approach has been very successful, time-triggered architectures [83] have exploited this idea to produce some very dependable real-time systems [154].

However, the major disadvantages of static systems are that they lack the flexibility to react to unusual or sporadic events and they cannot easily adapt to changes [82]. Further, the increasingly dynamic functional requirements are hard to support on a

largely static system; flexibility might be vital to support particular applications.

The meaning of *flexibility* will be explored in more detail in the next chapter, but in the context of this work, the word flexibility can be considered to mean the ability to make decisions at run-time about which node may transmit at a given time. Controller area network (CAN) [28, 72] is a flexible, 'event-based' bus system where messages are scheduled on the bus *dynamically* at run-time using priority-based arbitration. CAN is prevalent in embedded systems, particularly in the automotive industry. CAN currently holds a massive market share—in the year 2000 over 10^8 CAN nodes were sold [89]. It is for this reason, together with the technical merits of the protocol (described in Chapter 2), that CAN was chosen as a basis for this research. Much of the work in this thesis uses CAN as an example of a flexible bus and includes CAN-specific details.

Within the time frame of the research presented in this thesis, the automotive industry (which has been the major driving force behind the CAN market) has been considering new bus systems to replace CAN in future years. As the applications become more and more safety-critical, time-triggered architectures seem to be winning the debate [111] on the grounds of timing predictability. The two major contenders are [110] the well established time-triggered protocol (TTP) [82] and the newer FlexRay [19] which is based largely on a time-triggered paradigm but incorporating some event-based/dynamic communication.

1.4 Real-time and Control Systems

There are many forms of distributed systems; this thesis focuses on *control systems*, characterised by the idea that the system is controlling or interacting with some real-world objects. Examples of control systems include: large scale factory automation where many machines—distributed widely—are under some combined coordination; small scale automation such as computer control of a lathe, a milling machine or robotic 'arm'; vehicle management; aircraft control and so on.

Automotive examples such as coordinated engine management, assisted braking and gearbox control are very familiar [147, 114]. The automotive industry has been

a major consumer of distributed technology, extensively using bus technology to link *smart sensors* to computers all around the vehicle.

Since control systems are concerned with controlling real-world objects, such systems usually have inherent timing constraints dictated in part by the physical characteristics of the objects they control. Therefore, a vital property of the communications subsystem is to be able to support these timing constraints in communication. It is not obvious however, what form this 'support' should take. It is widely considered, especially in real-time systems research, that bounded latency is a vitally important property for real-time communication [159, 131]. *Deadlines* are also often used to describe timing constraints and infer correct behaviour.

However, engineers will confirm that these are only a few of the many concerns including bandwidth, fault tolerance, reliability, cost, size, interfacing and the provision of other services like clock synchronisation and membership. For example, most digital feedback control loops benefit from being executed *periodically* with minimal variation [105, 10]. Most of these concerns have mutually exclusive implications and can only be resolved through tradeoffs.

1.5 Contribution

This thesis argues that in the context of embedded computer communication, the *flexibility* of an event-triggered bus can be *exploited* to provide *reliability*. Further, in an unpredictable environment, dynamic behaviour is an effective means of providing fault tolerance.

The first point of contention is that guarantees based on meeting all deadlines are not only *impractical*, but in some situations are *impossible*, no matter which form of communication is used. However, it is not *necessary* to meet *all* communication deadlines in a dependable system.

Further, this thesis shows that in flexible communication, allowing a small number of deadlines to be missed *in a predictable way* can result in a communication system which is reliable, tolerant to faults and capable of supporting dependable real-time systems.

The major contributions of this thesis are:

Weakly Hard Analysis of CAN. The application of *weakly-hard analysis* to CAN is used to predict which deadlines may be missed and how often. With some fault models, weakly-hard analysis allows one to state that for example: “in any 20 consecutive messages there will never be more than 3 deadlines missed, and there will never be two consecutive missed deadlines”. This topic is covered in Chapter 3.

Exploitation of Non-Harmonic Periodicity. In a traditional cyclicly scheduled system, it has been necessary to ensure that periodic messages have a period which is a multiple of the ‘bus cycle time’. In a flexible system such as CAN not only is this not required, but there are significant advantages to fault-tolerance if the periods are largely non-harmonic. This subject is also described in Chapter 3.

Probabilistic Distribution of Response Times. The introduction of a more realistic fault model based on a probabilistic approach—where faults occur randomly—leads to an analysis which gives a probabilistic distribution of response times. This in turn gives a probability of timing failure. This model and analysis is explained in Chapter 4.

Predictable Failure. Given that analyses exist to predict which deadlines may be missed and how likely they are to be missed, Chapter 5 introduces an extension to CAN, called TCAN (Timely-CAN), which allows the communication system to be aware of time as a first class entity, and to use information about deadlines to selectively transmit frames. The result is a predictable bus which under high levels of unpredictable, environment-induced faults will behave more reliably than than CAN alone is able to.

In addition to these main chapters, a review of relevant literature is given in Chapter 2, and an evaluation of the work presented, with conclusions and a description of further work is given in Chapter 6.

2 Real-time Communication

FLEXIBILITY IN REAL-TIME COMMUNICATION is the central theme to this thesis. In this chapter, relevant background material is presented, including a discussion about what flexibility means in real-time communication and how faults and unpredictable behaviour can be dealt with. A well known bus protocol, Controller Area Network (CAN), will be discussed in more detail. The chapter begins with brief introductions to real-time computing, communication and dependability.

2.1 Real-time Systems

The first chapter introduced a *real-time* system as one where "the times at which it interacts with its environment are important in some way". There are numerous definitions of the term "real-time", but this broad explanation is perhaps the most useful interpretation of real-time systems. This section describes relevant theory from real-time systems research. Various tools and techniques that will be used later in this thesis are described. In particular, the concept of a deadline will be introduced as a way of describing real-time communication.

2.1.1 On Definitions and Classification

It is perhaps inherent in the psyche of computer scientists and mathematicians to attempt to classify and label according to some exact calculus of characteristics. Many times in the past, people have tried to further define what "real-time" means, and there are a wide variety[1] of over-argued[2] answers to the question.

[1] See http://www.faqs.org/faqs/realtime-computing/faq/ for some.
[2] See news:comp.realtime.

Yet, to attempt to construct a well-defined boundary around the set of systems that we call "real-time" is both as fruitless and as pointless as trying to decide if grey is black or white. Instead, it is sufficient to consider that *time* is important *in some way* to the system in its interaction with its environment.

Furthermore, for the large number of classifications within real-time systems, it is neither possible, nor necessarily useful to attempt to produce precise definitions of sub-classes of real-time systems. These boundaries are as unclear as the boundary between systems that we call real-time and those that we do not.

However, there is significant merit in describing formalisms that enable practical real-time systems to be constructed. These formalisms should be viewed as 'tools' that the real-time systems designer has at his/her disposal. Tools such as *deadlines*, *hard deadlines*, *soft deadlines*, *response time analysis* and so on are the means by which real-time systems can be built.

So, instead of striving for precise definitions and classifications of systems, the following sections explain a number of the more useful *techniques* that may be used to describe a real-time system and how a system interacts with its environment with respect to time. It is the "*in some way*" that is formalised here.

2.1.2 Correctness

For many systems, it can be said that the correctness of a real-time system depends not only on the logical result of computation, but also on the time at which the results are produced [39]. This applies to many real-time systems, not least control systems where the *raison d'être* is to control some physical object such as the fuel input to a car engine. If the computer control fails to activate the mechanics of the valves at the right times, then the engine will not work correctly.

For many real-time systems, it is difficult to determine what 'correctness' means. Significant real-time systems research involves improving *performance*, where the boundary between poor performance and incorrect behaviour is not clear. For example, in video decoding in consumer electronics, a temporary drop in performance may lead to a poorer picture quality, but whether or not this is incorrect behaviour is a very

subjective question.

2.1.3 Deadlines as a Tool

Much of the early research in the area of real-time systems [103, 134, 95] focused on what are now called *hard deadlines* and recognised to be one of the easiest forms of real-time behaviour to reason about (although not necessarily the easiest to construct).

A deadline is a boundary point that marks the latest time that an event should occur. Deadlines are a useful tool because they map time from a continuum into only two regions (before or on the deadline and after the deadline); this makes reasoning about behaviour much easier and allows a variety of formal specification and model-checking techniques to be applied such as RTL (Real-time Logic) [76], CML (Concurrent ML) [133, 135] or Timed Automata [6, 37].

In particular, a *hard deadline* implies that the behaviour of the system is incorrect if the deadline is missed and correct otherwise. For example, consider a robot arm collecting items from a moving conveyer belt. There is a hard deadline on the time that the robot arm must grab an object: it must grab it before it is out of reach. If the robot arm fails to grab the item in time then the system is functioning incorrectly.

It should be stated that although it is normal to separate the consequences of failure from how 'hard' a deadline is, in practice there is a clear correlation between consequences of failure and the use of hard deadlines. Indeed, some have suggested that a deadline is not considered hard unless its consequences of failure are severe [12].

Hard deadlines are easy to reason about because they allow the creation of a precise boundary point between correct and incorrect behaviour. Therefore, provided sufficient information is known about the system, then timeliness (as defined by hard deadlines) may be proven or disproven with the same significance as the results of an algebraic calculation or algorithm may be proven. Further, timeliness may be tested easily during the lifetime of the system: a deadline was either met, or it wasn't; this allows testing procedures to be formalised and the justification of confidence in a system's correctness.

Hard deadlines are therefore a useful tool that may be used to describe a real-time

system. Section 2.6 will show how hard deadlines can be used in communication, and later in Chapter 3 a variation of hard deadlines called weakly-hard deadlines will be introduced.

2.1.4 How Hard is a Hard Real-time System?

A system where there are activities with a hard deadline is often referred to as a *hard real-time system*. Hard real-time systems can be easy to understand and to reason about because of the advantages of using 'hard deadlines' to provide a precise mathematical foundation.

Yet, one of the problems with hard deadlines is that they may lead to an over-constrained system, difficult and expensive to implement because the demands placed by hard deadlines are difficult to meet. To design a system which does not have to meet *every* deadline allows simpler, smaller systems, which make better use of resources [22].

In the real-world, failures happen. Even the most well-designed, over-engineered system can suffer a failure; the effects of EMI, high-energy particles from space, hardware degradation *etc*. are difficult to eliminate completely. The best that can be achieved is some high level of confidence, perhaps expressed as a probability of failure. Timing failures are included in this discussion—the confidence in an 'absolute' deadline guarantee is not clear. So, questions arise such as 'is this really a hard deadline?' and 'how hard is hard?'.

Some partial answers to these questions in the context of real-time communication will be seen later. Systems may have many activities for which it is useful to specify a deadline, however there are fewer occasions where it is necessary for the deadline to be hard (and to do so would place unnecessary constraints on the design) [43, 21].

Chapter 4 will show that it is not possible to build a reliable communications system which is able to guarantee hard deadlines, because the effect of faults cannot be predicted. Instead probabilities will be used to predict deadline failures.

2.1.5 Softer Deadlines

To overcome the difficulties of implementing hard systems, the concept of a soft deadline is understood; it carries an implication that the deadline does not always have to be met.

However, once we move away from hard deadlines with their clear semantics, and move to the fuzzier concept of a deadline that doesn't have to be met, numerous difficulties arise when it comes to reasoning about behaviour. So, some formalisation is required in order to allow reasoning about system-level behaviour when deadlines are soft.

Various schemes exist for formalising 'soft' deadlines including firm deadlines, utility functions [99], weakly-hard deadlines [22], miss ratios, probabilistic guarantees [38], mean (or maximum) tardiness, summed tardiness, criticality-driven deadline failures [26] and so on. Certainly, different schemes have uses in different applications.

For some applications, a late result is better than no result, for example if there is some (small) delay in getting an instruction to apply a brake in a car, then it is better to brake late than not at all. On the other hand, a message that arrives after braking is no longer required could lead to catastrophic consequences.

For other applications, there is no utility in a late result: consider the robot arm example before, if the item is out of reach then there is no point in reaching anyway. The term *firm deadline* is often applied to a deadline for which a late result has no benefit. One important application of firm deadlines (which will be used later in this thesis) is in fault-tolerant systems with hard deadlines. Consider a system with hard deadlines—where the expectation is that all deadlines are met under normal circumstances—but to guarantee to meet all deadlines under unpredictable fault conditions is either difficult or not possible. Instead, the system is able to tolerate a (small) number of mmissed deadlines; therefore we consider these deadlines to be firm deadlines with additional constraints on the maximum number of tolerable timing failures.

2.1.6 Other Ways of Describing Real-time Systems

Deadlines are a very useful tool, however they are not the only useful way to describe timing. For some systems, it is important that events cannot happen too early. For some systems, keeping *variation* in timing to a minimum is important: minimal *jitter* on periodic sensor readings, for example, may be important for a control law to work efficiently.

Throughput is an alternative way of describing real-time behaviour without deadlines. As an example where throughput is more appropriate than a deadline, consider a real-time database. In order to maintain the desired service, it must handle (say) 100 requests per second. This is not the same as having a deadline per request of $\frac{1}{100}$ s since its requests are not all the same: some require more processing than others. Yet through knowledge of the "distribution" of requests, it can guarantee timeliness as defined by throughput.

2.2 Real-time Communication

The introductory chapter explained that the ability for a communication subsystem to support real-time behaviour (whatever that means in a given context) is desirable. The previous section introduced deadlines as one of the most useful tools for describing real-time systems. In computer communication, it is natural to assign deadlines to messages, as for events in any other real-time system, or indeed for messages in the non-computer world, "This letter must be delivered before 10 am tomorrow".

Although real-time computing has its roots in CPU scheduling [134] and there is a huge mass of research devoted to single processor and multiprocessor scheduling techniques, some of the ideas and concepts are directly transferable to communications. There is a powerful analogy between messages on a network or bus and jobs executing on a CPU. It is useful to regard a bus and a CPU as *shared resources* which must be shared between competing messages or jobs.

However, there is a major difference, with many practical implications, between real-time computing on a CPU and real-time communication.

The difference is '*knowledge*'. On a single CPU, the operating system which makes the scheduling decisions has complete knowledge of the current state of the system: which processes are ready to use the CPU, which processes are blocked on some resource, and so on. This allows the operating system to make good scheduling decisions and enforce a huge variety [136, 53] of scheduling policies.

However for a distributed system, in general no single node holds all of the same information. Instead this knowledge is distributed around the system, small parts of it on each node (each node knows which messages it wants to transmit next, but no other node knows this).

Distributed knowledge [58] is not sufficient to enforce even simple scheduling policies such as fixed priority based scheduling because no node knows whether it has the highest priority message. Some method is required to disseminate sufficient common knowledge around the system for a scheduling decision to be made. Numerous ingenious ways of doing this exist, although it is beyond the scope of this document to describe them in detail. Notable schemes for real-time communication include time division multiple access (TDMA), flexible time-triggered (FTT), bitwise arbitration (*e.g.* CAN) and non-deterministic (*e.g.* ethernet), each of which are described later in this chapter.

Clock synchronisation is a response to the special case of a lack of common knowledge about *time*. Each node only has access to its own internal clock, which may drift with respect to the other clocks in the system. Therefore, each node will have a (slightly) different interpretation of the time. In some systems, this may not matter at all; many event-triggered systems do not need a consistent global view of time. However, some systems do need time-based synchronisation in order for the application to function, therefore some means to deal with clock skew is required. Many clock synchronisation algorithms exist [7, 120] some of which are implemented as high level protocols, some can be implemented at a lower level as part of the network protocol itself [56, 84]. The specifics of clock synchronisation will not be discussed further here, but the topic will be covered briefly in later chapters.

There are a large number of choices facing a designer when choosing a communications bus. Some well-known fieldbuses used in real-time control systems include:

BITBUS [71], Interbus-S [68], LONWORKS [47], PROFIBUS [130], and WorldFIP [15]. A large list including over 300 different fieldbuses and higher level fieldbus protocols has been compiled [67]. Nearly all of them may be used, in some form, in a real-time control system.

For some communication systems, the ability to support real-time communication has been key from inception: TTP (Time-Triggered Protocol) [163], described shortly, provides direct support for timely delivery of messages, clock synchronisation and several other services. For other protocols, analysis of real-time properties has been considered only later, once the protocol had been used successfully. CAN (Controller Area Network) [28] is an example of this.

Even buses which are not generally regarded as able to support real-time systems, such as ethernet [70] (since collision resolution is non-deterministic) can be used for real-time communication in many systems. For ethernet, the bandwidth is so high compared to other buses, that for an equivalent traffic level, bus utilisation on ethernet is low enough that it outperforms many other "real-time" buses in terms of observed worst case response time and average latency.

2.3 Dependability

Leaving communication aside for a moment, it is useful to explore a little background information from the (very large) areas of *dependability* and *safety*, and to clarity some terminology.

Dependability, reliability, safety and similar words are often used to mean several related things. To avoid ambiguity, the widely accepted terminology of Laprie [88] will be used throughout the thesis wherever possible.

> *Dependability* is the trustworthiness of a computer system such that reliance can justifiably be placed on the service it delivers [42]. The *service* delivered by a system is its behaviour as it is perceived by its user(s) [88].

'Dependability' is therefore subjective, depending on the applications and the users.

In order to allow dependability to be considered formally, Laprie lists four *attributes* of dependability:

- with respect to *readiness for usage*, dependable means *available*;
- with respect to *continuity of service*, dependable means *reliable*;
- with respect to *avoidance of catastrophic consequences*, dependable means *safe*;
- with respect to *prevention of unauthorised access and/or handling of information*, dependable means *secure*.

In this thesis, concerning control systems, context dictates that the first three attributes are the most important. Despite security becoming more relevant nowadays, the possibility of malicious tampering with communications holds little concern since physical security may be generally be relied upon in vehicles, aircraft and factories. Instead, availability, reliability and safety are vital attributes of any avionics/automotive product, and therefore these attributes are required of sub-systems, especially the communication sub-system.

Given that dependability is based on correct delivery of services, a *failure* is said to have occurred when service is not delivered according to some *specification* or expectation of that service. The design of specifications is a difficult task though—specifications are unlikely to be absolutely correct themselves [48, 92].

Reliability requirements can be described in a number of ways. Typically, a probability of failure in a given time interval may be used. The Federal Aviation Authority rules require that a catastrophic failure condition (meaning loss or serious damage to the aircraft) must be "extremely improbable", defined as 10^{-9} per hour or "is not expected to occur within the total life span of the whole fleet of the model" [168].

The figure 10^{-9} is so small that, despite its well-understood mathematical definition, it may be difficult to comprehend its value and to apply it [92]. It is clear from the history of avionics accidents that catastrophic failures have 'occurred during the life span of the whole fleet of the model'. It is argued by some that the probability 10^{-9} per hour is so low that this level of reliability cannot be guaranteed or even measured [92]. On the other hand, the use of probabilities is useful; probabilities form the basis

of safety management through risk assessment. Defence standard 00-56 [108] defines *risk* as the combination of of the frequency, or probability, and the consequence of an accident. One difficulty with probabilities is their prediction—Leveson [93] reminds us that where probabilities are used, their value must be justified.

2.3.1 Fault→Error→Failure

It is useful to distinguish between a fault, an error and a failure [88]:

- a *failure* is said to have occurred when service is not delivered according to the specification or expectation of that service;
- an *error* is that part of the system state which is liable to lead to subsequent failure;
- a *fault* is the cause of an error.

A relevant example is illustrated, which highlights a fundamental issue that is addressed in this thesis: disturbances of the signals in a bus system. At the lowest level, a *fault* (such as some electrical interference, or a loose connection) causes some bits on a communications bus to be received incorrectly, therefore a data packet arrives corrupted (the *error*). This in turn causes the packet to be retransmitted, and when the packet finally arrives at the application, it is late (the *failure*).

These terms can be used recursively—fault, error and failure are applied in the same way that functionality is recursively decomposed. For example, from the perspective of the application that was due to receive the data packet, the *fault* is that the packet is late, the *error* is that it does not have the data in the packet and therefore the application's internal state is incorrect. The *failure* is that the application cannot perform its calculations correctly and so incorrectly adjusts an actuator on the wing of the aircraft.

At yet a higher level, from the perspective of the aeroplane just as it is taking off: the *fault* is that the actuator on the wing moves, the *error* is that the plane rolls sharply, and the *failure* is that the plane was in danger of catastrophe: a wing touching the ground.

It should be noted that at each of the levels above, one should expect to see some method of dealing with the faults and errors to prevent errors becoming failures. The bus in the example did try to get the message through by retransmission of the corrupt packet. However the packet still arrived late, causing subsequent failures elsewhere in the system. In the example, a single late message causing a catastrophic failure is unrealistic. It would be expected that the application should be able to deal with a late message through a variety of fault tolerant techniques.

2.4 Flexibility in Real-time Communication

Returning to communication, this section introduces the term *flexibility* and how it relates to real-time communication. The term has been used in this thesis so far without definition, indeed its use is difficult to define. Examples of flexibility in communications will be provided which will lead to an understanding of what flexibility means for real-time communication.

In any network/bus system, some mechanism is needed to determine which node may transmit at any given time. This mechanism is called a *protocol*. Specifically, when used for this role of node *arbitration* it is called a *medium access control* (MAC) protocol. A comprehensive survey of MAC protocols in real-time communication is provided by Malcolm [102]. A few groups of protocols and some specific protocols are examined in the following.

2.4.1 Time-Triggered Communication

To examine flexibility in communication, a group of bus systems may be considered based on TDMA (*Time Division Multiple Access*) protocols, where bus access is determined entirely by predefined time-slots. The most notable example of a TDMA bus system is TTP [163, 82], which exists in two forms known as TTP/C and a simpler form, TTP/A.

TDMA is a scheduling strategy whereby the right to transmit is controlled by the progression of time. Generally, TDMA schemes divide time into repeating *cycles*

(sometimes also into major cycles and smaller minor cycles), each node is allowed to write to the bus only at certain times within a cycle. In order to ensure that collisions cannot occur, each node is given a different time *slot* in which to transmit. The slots and cycles are determined off-line, often using tool support. The cycles are usually the same length and repeat indefinitely, making the bus *periodic* in nature. Figure 2.1 shows an example of TDMA in which two cycles are shown. Notice that message A appears more than once in each cycle.

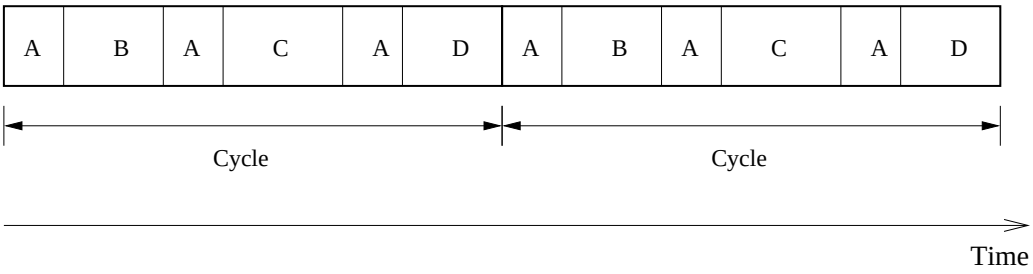


Figure 2.1: Two Cycles of a TDMA schedule example.

Time-driven TDMA communication is used in many forms, not just in fieldbus applications. In particular, TDMA is used extensively for radio communications, for example GSM mobile telephones use TDMA technology to multiplex a number of different transmitting telephones onto one radio frequency [132]. Other radio based communication protocols use similar techniques.

Fundamentally, with TDMA, all scheduling is determined in advance and therefore the scheduling information is available (*known*) to all nodes *a priori*: the schedule is common knowledge [58]. This means that the problem with distributed knowledge about scheduling that was highlighted in Section 2.2 is avoided in TDMA communications.

This *predictability* has some important and useful implications. Since all scheduling is determined in advance, real-time behaviour is very easy to reason about. It is easy to find worst case response times and show which deadlines will be met through off-line analysis tools [164]. For many systems, the key benefit of TDMA is the ability to provide guarantees (assuming no faults) of hard real-time deadlines in communication.

Additionally, the common knowledge of the schedule may be used to efficiently provide other services including addressing/identification and membership. TTP exploits TDMA to provide low overhead implementations of a global time service, membership services and distributed modular redundancy [82]. For addressing/identification, the transmitter (of a frame) is always known and so the data on the bus is always identifiable, therefore there is no need for any addressing information to be included in the frame (which has bandwidth savings over other schemes). For a membership service, if a node fails to transmit during its slot then all nodes will instantaneously know that the node has failed, hence providing a cheap and implicit implementation of a membership service. TDMA is also easy to implement, simple to understand and, importantly in a dependable real-time context, its behaviour is predictable.

The major disadvantage of TDMA is the difficulty in scheduling the unknown: where the *exact* communication requirements are not known in advance [82]. The difficulties are a consequence of requiring all scheduling information to be determined off-line. This is fine in systems where all communication requirements are known and static, but TDMA may not provide an efficient implementation for non-static systems. Examples include: open systems (where applications may dynamically join and leave the system), applications where the communication requirements are not precisely known, applications whose requirements may change in time, or applications with non-periodic network traffic.

One area in particular where TDMA schemes may struggle is when there is corruption on the bus. Consider a fault which causes some bits on the bus to be corrupted, hence the frame in which they were transmitted is lost. If there is to be any chance of receiving that data correctly then there must necessarily be some form of redundancy: for example by using an error correcting code [59] or by routinely retransmitting frames. The approach to frame redundancy taken by TTP is to send all frames twice [163]. This halves the available network bandwidth to approximately double the probability that at least one copy of the data will arrive intact. Of course, if the second frame is also corrupted then the frame is certainly lost.

What is lacking from TDMA is the *flexibility* to be able to make bus scheduling decisions at run-time, this would allow selective retransmission of corrupt data to provide a more efficient error recovery mechanism. Of course if scheduling decisions

are made *dynamically* then the common knowledge of the schedule is immediately lost; some other means must be used to share sufficient (distributed) knowledge around the system for scheduling decisions to be made. A variety of buses exist which do allow for more flexibility, some of which are considered next.

2.4.2 Introducing Flexibility—ARINC-629

ARINC-629 [8] is a bus designed specifically for modular avionics systems [1], and as such must provide support for hard real-time deadlines as well as predictable, certifiable behaviour and fault tolerance. ARINC-629 is now used extensively in avionics, notably the Boeing 777 uses 629 for many safety critical functions including control [20].

Inherent in the design of ARINC-629 was that it should allow a system using the bus to be upgraded or modified easily—this requires a certain degree of flexibility. Further, it must be capable of supporting the wide variety of different forms of message that exist (and were forecast to exist when the bus was designed) in both military and civil aircraft. This data includes regular *periodic* flight control data (including allowing the period to change whilst in flight depending on the current operational mode of the aircraft) and sporadic flight management system information [152].

The access resolution mechanism of ARINC-629 is loosely based on time slots and therefore it has much in common with TDMA protocols. However, it also contains explicit support for sporadic information.

The problem of distributing sufficient knowledge to make scheduling decisions was solved in TDMA by disseminating it off-line. Where this is not possible (because it is not known for sporadic information), then some on-line mechanism is required to arbitrate between nodes that wish to transmit at the same time. ARINC-629 is a member of a class of MAC protocols known as CSMA (Carrier Sense Multiple Access), which are characterised by arbitration decisions being made on the basis of the bus status (and local information). Typically in a CSMA protocol, all nodes wishing to transmit wait until the bus is silent, *e.g.* at the end of a packet/frame (carrier sense), then compete in some arbitration scheme.

Arbitration in ARINC-629 is handled by *collision avoidance* (CSMA-CA). A set of timers in each node ensures that when the bus is silent, each node waits a different amount of time before attempting transmission.

ARINC-629 has two protocol modes, the *basic protocol* and the *combined protocol*, of which the combined protocol is the most suited to mixing periodic and non-periodic traffic. The overall bus scheduling scheme is first to transmit the periodic messages in TDMA style (although using a slightly unusual scheme to implement this,[3]) then to allow sporadic messages to compete for the spare bandwidth at the end of the minor cycle. A timing analysis for ARINC 629 [13] shows that the protocol is capable of supporting periodic traffic with deadlines, and sporadic traffic with deadlines, provided that the worst case sporadic traffic in the system is known.

So, ARINC-629 is similar to TDMA with room for non-periodic data at the end of each cycle after the periodic messages are safely transmitted. Similar approaches to scheduling periodic and non-periodic information are taken by ProfiBUS [130] where the implementation is based on token passing, and WorldFiP [15] which is based on a master node performing centralised control. There is some *flexibility* here because the decision about which message to transmit is taken at *run-time* by the competing nodes.

However, ARINC-629 stops short of providing any significant fault-tolerance support against faults introduced onto the network by interference. It simply leaves all such concerns to the application, except for the provision of a single parity bit. The major fault tolerance feature of ARINC-629 is the expectation that there will be multiple redundant buses. Further, it is expected that applications must be able to tolerate a small number of network faults (such as corrupted messages) which result in message omissions. ARINC-629's 'firm deadline' model of delivery is that messages arrive on-time (with a high probability) or they do not arrive at all. This model of using only firm deadlines, as mentioned in Section 2.1.5, is useful and opens several opportunities which will be discussed in later chapters.

This raises the question: at what level should reliability and fault tolerance be handled [62]? In general, infrastructures which provide reliable communication services

[3]The mechanism is based on lengths of idle time following each frame, see the specification [8] for full details.

tend to be inherently complex and consume significant resources (such as bandwidth). However, providing fault tolerance at the low-level can be implemented efficiently and it is clear from an engineering perspective that it may lead to highly reliable systems. On the other hand, the approach used by ARINC-629 (pushing fault tolerance higher up in the system) can also lead to high reliability systems. Implementing fault-tolerance mainly at the application level can also lead to efficiency advantages since the fault tolerance 'level' can be tuned to each application by exploiting the natural robustness of the applications, but this comes at the cost of application complexity and a great deal more effort than using a 'ready made' reliable infrastructure.

2.4.3 Flexibility through Centralised Control—FTT

Almeida [3] proposes a centralised scheduling scheme where a central master node, the *planning scheduler*, creates a TDMA schedule and informs all the other nodes about it through a broadcast message. FTT stands for *flexible time-triggered*. The flexibility comes through the ability for nodes to request changes to the schedule *at run-time*. The master node can then try to determine a TDMA schedule which can accommodate the new request.

An advantage of centralised control is that the scheduler can be aware of the requirements of all nodes in the system (common knowledge) and can then attempt to derive an optimal schedule to meet those requirements. On the other hand, there is potential for a large overhead in distributing requests for bandwidth to the master and then for the master periodically to distribute schedules.

The centralised planning scheduler can be regarded as a higher level concept than a network protocol; the FTT scheme has been applied to a number of networks including Ethernet [122], CAN [4, 5, 2] and WorldFIP [2].

2.4.4 Flexibility—802.3 (Ethernet)

In complete contrast to a TDMA bus (where all scheduling decisions are determined off-line) and ARINC-629 (where some scheduling decisions are taken off-line), a class of protocols may be considered which are often called *event-triggered* protocols, char-

acterised by all scheduling decisions being taken at run-time. The advantage of delaying scheduling decisions until run-time is that the bus is very easily able to handle sporadic data, or even unpredictable events such as faults.

The most well known event-triggered network is Ethernet [70]. Here, (in normal half-duplex mode) there is a non-deterministic access resolution protocol: if two nodes wish to transmit a packet to the bus at the same time, then this is detected (CSMA-CD, *collision detection*) and they both stop transmitting and wait for a random time. Therefore it is a random choice which of them (or any other node) transmits first. For general purpose computing, this works remarkably well, especially at low–medium bandwidth levels, although if demand is high then Ethernet struggles to cope (the maximum useful utilisation limit of ethernet depends on many things and is not straightforward to evaluate [27]). Switched Ethernet (which employs point-to-point full-duplex connections between nodes and a central 'switch') copes better with overload, but requires a complex, potentially expensive, central switch to manage the network and involves considerable wiring between the switch and the nodes.

Ethernet provides very little fault-tolerance for packets of data. Like ARINC-629, Ethernet leaves error correction to higher level protocols like TCP/IP through retransmission of lost packets. This arrangement works quite well in practice, although overload scenarios can lead to long delays.

The non-deterministic collision resolution is the main reason why Ethernet is not generally used for real-time communication. Even if the traffic is known in advance, if there are collisions then it is not possible to state the worst case latency of any packet. Therefore, deadline compliance is difficult to predict. Despite this, a number of techniques may be used to provide some level of real-time predictability: token-passing protocols may be used to avoid collisions, a TDMA policy may be enforced, FTT-Ethernet does this for example. Alternatively probabilistic analysis techniques may be used with success, especially when combined with traffic shaping [85, 41].

2.4.5 Flexibility for Fault Tolerance—CAN

There are many different types of flexible/event-triggered networks. Of particular interest is the CAN bus protocol [28], described in more detail in Section 2.5. CAN, like Ethernet, uses a collision detection protocol (CSMA-CD) but unlike Ethernet it is a *deterministic collision resolution* (CSMA-DCR) scheme[4] meaning that when two nodes wish to transmit at the same time, it is always possible to predict the one that actually transmits first.

Furthermore, the protocol provides *non-destructive collision resolution*: when a collision occurs between two or more frames, the frame with the highest priority continues transmitting and any other nodes stop transmitting their lower priority frames (the means by which this is achieved is explained in Section 2.5.4). Therefore, CAN provides a non-preemptive, priority-based bus scheduling protocol. Importantly, worst case response times can be calculated [159] to provide deadline guarantees.

Unlike TDMA schemes, ARINC-629 and Ethernet, CAN provides significant fault-tolerant features. In particular, CAN tries to ensure *reliable frame delivery* by automatically retransmitting any frames that were corrupted on the network. The error correction is done as part of the network protocol itself, rather than relying on higher-level protocols (such as TCP/IP) to provide a reliable service.

CAN therefore uses its flexibility in a useful way, through the use of fault tolerance, to implement a reliable service. It is able to do this because it makes all scheduling decisions dynamically at run-time.

As highlighted previously, CAN has some interesting and useful properties for distributed real-time systems, particularly the network-level fault tolerance. In addition to these, its massive market share and its high-standing position in industry and academia mean that CAN research has a very wide audience. The next section presents CAN in more detail; it will show that CAN has analysable timing properties and can provide flexible, fault tolerant communication.

However, despite these properties, the role of CAN in future dependable systems is undecided; there is a lack of understanding about timing analysis on CAN and there is

[4] The CAN MAC protocol has also been classified as CSMA-UID (unilateral interface detection) [87] and CSMA-PCR (priority collision resolution) [3].

a recent move to implement a time-triggered CAN (TTCAN) [52] in order to improve timing reliability in dependable systems. Yet, as this thesis will demonstrate in later chapters, the flexibility of CAN alone may be used to provide reliable real-time communication, without resorting to a time-triggered paradigm. New analysis techniques can provide better insight than previous forms of worst case analysis. And in Chapter 5 a small extension to CAN is proposed which provides similar timing predictability to TTCAN, but which may outperform TTCAN in term of fault tolerance.

2.5 Controller Area Network

Controller Area Network is a very well-known bus system that is widely used in industry for communication in small-scale distributed systems. As Chapter 1 mentioned, a major CAN user is the automotive industry, and in the year 2000, there were 10^8 CAN nodes sold worldwide [89]. Particular attention is paid to CAN partly because of its popularity, but also because of its technical merit as a flexible real-time communications bus. In this section the Controller Area Network protocol (CAN) will be described in some detail. CAN will be used throughout this thesis, leading eventually to a proposed extension to CAN in Chapter 5.

2.5.1 CAN, a Brief History

CAN is a serial communications bus which has many desirable properties for use in real-time dependable systems. Its specification was first published in 1991 by Bosch [28] and standardised by ISO in 1993 [72]. It was originally intended for use in vehicles as a single digital bus to replace the wiring harnesses that were growing in complexity as new electrical and electronic appliances appeared in cars.

The advantages of a single bus over a wiring harness are numerous: easier to install, lighter, cheaper, simpler to maintain, easier fault diagnosis, extensibility (more devices can be simply plugged in to the bus) and so on. The cost-conscious automotive industry quickly made CAN the *de facto* standard for automotive electronics.

In this environment, the data that is transmitted around the vehicle is mainly small

messages, perhaps one or two bytes for wheel speeds and actuator control data, binary values for switch positions. These are typically sent periodically, with fairly soft real-time specifications. Such applications have driven the motivation for CAN's relatively short data frame length (up to eight data bytes per frame). Data rates on CAN are also quite low, usually ranging between 125 kbit/s and 1 Mbit/s. A low bandwidth of 125 kbit/s may often be preferred because at lower rates, not only a bus is less prone to electrical interference [149], but because of the additional support of the physical layer, as described later in Section 2.5.3.

CAN's successful history in the automotive industry as a flexible embedded bus has driven its use in other applications such as factory-control networking, elevator control systems, machine tools, photocopiers and many more. The protocol has become readily available, nowadays there are many commercial hardware implementations of CAN providing numerous low-cost options, including single chip microcontrollers with in-built CAN interfaces, for example the Motorola M68376 [109]. The advantages of using a bus such as CAN in these applications may include: reduced design time (readily available components and tools) and lower connection costs (lighter, smaller cables and connectors).[5]

At its conception, CAN started out with little emphasis on real-time properties; the response time analyses were developed later and independently [159]. Instead, the aim of CAN was to provide a simple, efficient, robust communication system.

The emerging applications of CAN (both in-vehicle and elsewhere) began to impose real-time requirements of the bus which CAN was generally able to support. For example, in a vehicle fuel control application, message transmission requires short, hard/firm guarantees on message transfer latency.

By the mid-1990s, products based on CAN were proving to be reliable, with low failure rates and easy fault diagnosis. The next logical step was to consider the use of CAN in safety-critical applications such as anti-lock braking systems [115], drive-by-wire and even avionics applications. However, despite the mounting historical product-based evidence of reliability, and the acceptance by the avionics industry that CAN is largely suitable [9], using CAN as a vital link in a safety-critical system is a

[5] See http://www.mjschofield.com/

big step forward. This is partly because of a general uncertainty about using event-triggered systems to provide critical services, partly because of an unwillingness to certify something new [9], and partly because of a number of known shortcomings with the protocol [140], including the possibility of undetected errors, the error passive mode and inconsistent message delivery, see Sections 2.5.5, 2.5.6 and 2.5.7.

At the present day, CAN is being trialed in aircraft engines for non-critical services such as logging and maintenance. NASA has also shown interest in CAN for space applications [40]. Meanwhile, CAN continues to be used as a cheap, reliable bus in vehicle, factory and many other automation systems. The automotive industry, fuelled by the success of CAN, is now looking for the next generation bus system. The new protocol will have to fulfil the current role of CAN and in addition provide higher bandwidth as well as the ability to support some dependability requirements and flexible application demands.

CAN has a large market, a good history and a great deal of technical merit. It is for these reasons that in this thesis, CAN will be used as a specific example of an event-triggered protocol.

2.5.2 Layers of the CAN Protocol

The CAN protocol itself is not described here in any great depth; the reader should refer to one of the CAN specification documents for a clear [28] or more detailed [72] description. Nevertheless, in this document, an introduction to CAN is presented which should give the reader sufficient knowledge of its workings to understand the later material. Several specific aspects of CAN are discussed in more detail.

CAN may be viewed in terms of the OSI (Open Systems Interconnection) reference model [73, 161], providing three layers of the stack: the physical layer, the data-link layer and optionally an additional application layer, see Figure 2.2. The original Bosch standard [28] only specified the data-link layer, although a physical layer is specified by the ISO specification ISO-11898 [72]. Various higher-level protocols such as CAL/CANOpen [45], CAN Kingdom [51], CORBA and some protocols to ensure atomic broadcast (see Section 2.5.7) are also used to some extent. Rarely

are further layers of the OSI model used, the properties of the data-link layer being sufficient for many distributed control applications, using only a thin application layer. In this thesis, the various application layers will only be briefly considered as the physical and data-link layers are of more concern.

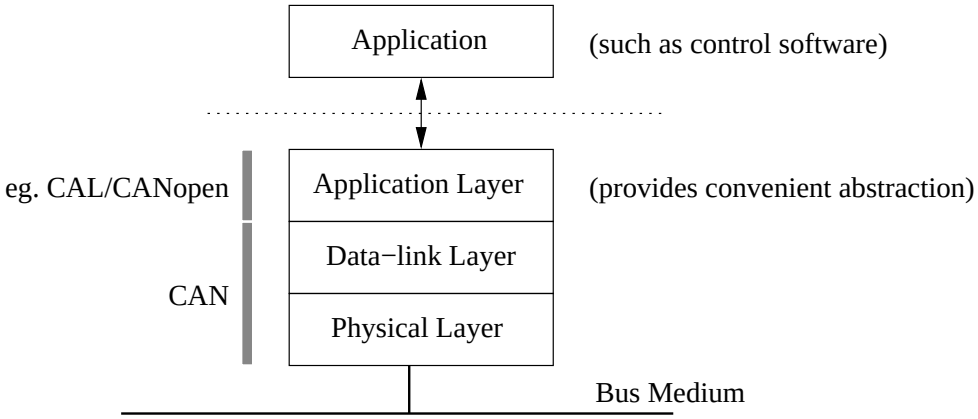


Figure 2.2: CAN Protocol Layers.

2.5.3 The Physical Layer

The physical layer of a CAN bus was originally not specified by Bosch in order that implementations were not tied to a particular medium (*e.g.* electrical cable, fibre optic *etc.*). Nevertheless, there is a small number of (largely similar) specifications for the physical layer including:

- *ISO-11898* [72] defines a two-wire balanced differential signalling scheme and NRZ (non-return-to-zero) coding at up to 1 Mbit/s, intended for general purpose, higher bandwidth applications;
- *ISO-11519* [74] defines a lower speed two-wire balanced differential signalling scheme at up to 125 kbit/s, intended for dependable and low bandwidth applications;
- several modifications of RS485 exist, suitable for CAN, which use higher voltages to offer more immunity to electrical noise;

- SAE-J2411 [148] defines a single wire physical layer intended for vehicle applications;

- an implementation using optical-fibres (for noise immunity) has been suggested [139].

ISO-11898 is the most widely recognised standard; most CAN transceivers (such as the Philips PCA82C250) implement this. The ISO standard will be assumed for all following discussions. Further details on the characteristics of the physical layer may be found in work by Rufino [140, 167].

The main property that the physical layer must provide is a 2-state medium; the two states are termed 'dominant' and 'recessive'. If two nodes simultaneously transmit bits of opposite value, then all nodes should read 'dominant'. It is usual for the dominant state to be associated with the binary value '0' and recessive with the binary value '1'. This is usually implemented by using a *wired-and* scheme: all nodes must transmit a '1' in order for the bus state to be '1'. This feature is heavily used in higher layers for medium access control and for error signalling.

The dominant/recessive coding also drives the limits on bandwidth in CAN. Any node must be able to overwrite a recessive bit with a dominant bit, and the original transmitter of that bit must be able to detect the change. Therefore, the duration of each bit must be long enough for the the signal to travel one complete length of the bus (the propagation delay). Hence, the maximum data-rate depends on the length of the bus, 1 Mbit/s is possible up to 40 m, and proportionally a 500 m bus demands a maximum data rate of 125 kbit/s.

The physical layer has a number of fault-tolerant features, which are very important in providing a reliable service. In particular, CAN provides resilience against a variety of physical faults such as one-wire being interrupted (broken) the two signal wires being connected together in a short-circuit, or one of the signal wires shorted to ground or power lines. These features are implemented by switching from differential signalling to single line signalling. Note that not all CAN controllers support this at higher bus speeds, for example the frequently used fault-tolerant Philips TJA1054 family [125] only supports data rates to 125 kbit/s.

Interference and the Physical Layer

The physical layer of a CAN bus (ISO-11898) is highly resistant to electromagnetic interference (EMI)—this was one of its design requirements. The differential mode signalling, in particular, is used to achieve this. Differential signalling is where two wires are used to carry the signal, usually with opposite voltages being applied to the two conductors. The actual signal is the (voltage) difference between them. Differential signalling is largely immune to external electromagnetic interference (EMI) since interference will tend to affect each side of a differential signal almost equally. Most noise occurring equally on the two conductors will therefore be ignored when the difference between the conductors is measured [66].

Despite these measures, however, there is a chance that EMI will affect the signal on the bus such that one or more nodes may simultaneously read a bit from the bus as a different value to that which was transmitted. This set of nodes may include the transmitter, in which case the error is detected immediately (since nodes will simultaneously read the bus when they are writing to verify that the value is written correctly). If the transmitter does not detect the error then it will probably be detected later at a higher layer of the CAN protocol, either by the cyclic redundancy check (CRC) or by one of the other measures in the data-link layer. Once a fault is detected, the frame is lost.

How many faults due to interference are there likely to be? How good is the physical layer at preventing EMI causing lost frames? Unfortunately, these questions are very difficult to answer in the general case. Each specific network with different cables, lengths of cable, numbers of nodes, types of drivers, screening, cable routing and so on will have a different response to EMI. Furthermore, it is difficult to *predict* EMI [86] as its causes are diverse (sparks, lightning, digital signals, radar, mobile phones [63], high voltage switching *etc.*) and uncertain (what is the worst/average effect of a lightning strike?).

Currently, the only real way to measure the effect of EMI is to monitor the actual system *in situ*, and even then there is no guarantee that the measurements are indicative of the future behaviour, or that the worst case has been measured.

There is evidence that, in general, it cannot be assumed that there are no faults due

to EMI. During electromagnetic compatibility testing of a car using CAN, error frames were monitored when high levels of radio frequency radiation were present [107]. The findings for that particular experiment were that:

- only certain small frequency ranges provided noticeable effects (corrupted bits), while interference over most of the spectrum had no effect whatsoever;
- and that screened cable and unscreened cable had similar performance.

There is no reason to assume that these findings apply universally to all CAN systems. Indeed, this is contrary to the expectation that in general screened cables provide far better shielding effect than unscreened twisted pair cable. However, wherever there is a long length of cable and a potentially noisy environment, there will always be the possibility of interference.

A conclusion to this brief discussion is that all electrical communication systems are susceptible to interference and although one can measure interference and estimate its effects in the future, the science is not exact and no guarantees can be made about the likely interference during the lifetime of the system.

2.5.4 The Data-Link Layer

Above the physical layer, the main properties of CAN are provided by the data-link layer. Traffic on the bus consists of packets (frames). There are four types of message on CAN: data frames, error frames, remote transmission request frames and overload frames. This thesis is only concerned with data frames and error frames as the other two frames are rarely used in practice.

The CAN 2.0 specification [28] describes two versions of the protocol: CAN2.0A and CAN2.0B. The differences between them are concerned mainly with a longer arbitration field (see below). Although CAN2.0B has uses, CAN2.0A is sufficient for most distributed control systems; in this thesis CAN2.0A will be assumed except where otherwise stated. The reason for the evolution of version 2.0B, as given by Bosch [28], is to allow structured naming schemes to become standardised in certain

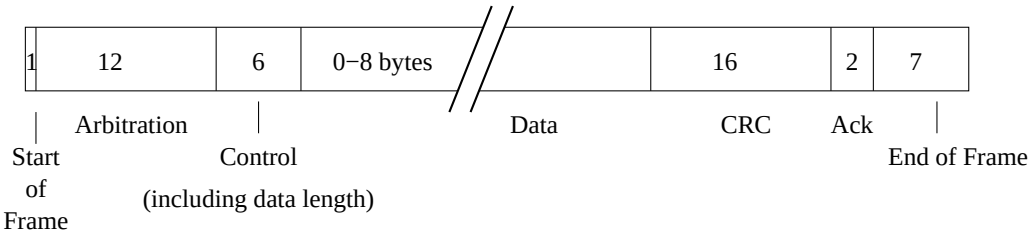


Figure 2.3: CAN Data Frame Format (simplified)—field sizes in bits except where stated.

applications. This appears to be driven by a desire for some level of compatibility with an alternative protocol used in American automotive applications.[6]

Data on the bus is sent in *data frames* which consist of up to 8 bytes of data plus a header and a footer. The frame is structured as a number of fields, as shown in Figure 2.3 and explained in the following.

Start	1 bit	A single dominant bit which provides a synchronisation point for all nodes on the bus
Arbitration	12 bits	This field contains the 11 bit priority identifier of the frame, used for both bus arbitration and for identifying messages. This is discussed in more detail later in this section. The additional bit is to indicate a *Remote Transmission Request* which is rarely used in practice and is not used for any further work in this thesis. CAN2.0B uses a 29 bit identifier field.
Control	6 bits	This field contains a four bit code to indicate how many bytes of data are to follow, and two unused/reserved bits.
Data	0–8 bytes	Up to eight bytes of application data can be sent. A zero-length frame is valid and useful in some applications, such as indicating the occurrence of events.

[6] This information is derived from http://www.algonet.se/~staffann/developer/CAN.htm accessed August 2003.

CRC	16 bits	This contains a fifteen bit cyclic redundancy check for the frame, plus one bit delimiter.
ACK	2 bits	Transmitters send the first ACK bit as recessive, receivers overwrite this with a dominant bit to indicate that at least one node has received the frame.
End	7 bits	7 recessive bits are transmitted to allow other nodes to signal an error in this frame. See the later discussion about Bit Stuffing.

Additionally, following the end of a frame, there must be a period of three bits of idle activity (recessive bits) on the bus, the *inter-frame space* during which all nodes may synchronise.

Arbitration, Identification and Addressing

The arbitration field serves three purposes: arbitration, identification and addressing.

The 11 bit field serves as a 'priority' for scheduling decisions. When more than one node on the bus wishes to transmit data, they monitor the bus waiting for it to become idle. At the earliest opportunity, all nodes with data to send will simultaneously transmit a start of frame bit. This is a dominant bit, so the value received by all nodes on the bus is dominant. Then each node wishing to participate will transmit the arbitration field of the frame, which is an 11 bit binary value transmitted most significant bit first. If at any time during arbitration, a node transmits a recessive bit, but detects a dominant bit on the bus, then it knows that there is a higher priority frame competing for arbitration. The lower priority node then stops transmitting, allowing any further higher priority nodes to continue.

By the end of the arbitration field, there should only be one remaining node transmitting (the node with the highest priority[7]) all others should have pulled out, so the

[7] Actually, the lowest arbitration value. In CAN, the lower the value in the arbitration field, the higher its priority. In common with much real-time literature, this thesis will adopt the convention that when quoting priorities a higher value implies a higher priority.

remaining node may continue to transmit its data. This is the means by which CAN provides *deterministic* collision resolution.

Once a data frame has started transmission, it will continue until it is completely sent or an error is detected. Nodes with lower priority frames will wait until the next idle tick on the bus and then try again to send their frames, competing with other nodes in the same way.

In CAN, there is no direct concept of *address*, familiar in protocols such as IP, nor a concept of message identification as used in TCP. Addressing and identification are handled indirectly using the arbitration field, so the second purpose of the arbitration field is identification. A stream of frames from a node (*e.g.* periodic sensor data) will commonly have the same priority/arbitration field. All receivers that make use of this data will know the arbitration value, and therefore when a field with a matching arbitration value is received, the semantic value of the data is understood.

The third purpose of the arbitration field is addressing. There are two ways that this is done: implicitly or explicitly. Typically, a node is only interested in certain data, which it recognises by the arbitration field. As CAN is a broadcast bus (all nodes receive all frames), by filtering out all frames except the ones with a matching arbitration field, CAN has an implicit addressing scheme. To facilitate this, a typical implementation of CAN [124] would allow an application to set a hardware mask which is compared against the arbitration field.

A more explicit form of addressing is also possible and is commonly used. The scheme breaks the arbitration field into two parts: the most significant bits used for priority as normal, but the lowest significant bits can be used as an address field. The highest priority bits determine the order in which frames are transmitted, but receiving nodes may set a mask to listen only for frames where the lowest significant bits match its own identifier (address). If such a scheme is used, the 29-bit identifier of CAN2.0B may be useful.

Bit Stuffing and Error Signalling

CAN defines a *bit stuffing* rule: during normal transmission of most[8] of a data frame, there should not be 6 consecutive bits of the same value (dominant or recessive) on the bus. Therefore, whenever a transmitter wishes to transmit 5 bits of the same value (including stuff bits), a *stuff bit* of the opposite value is automatically inserted into the stream. Naturally, receivers automatically remove any stuff bits from the stream before handing data upwards to the application. The effect of this on the size of frames is described in Section 2.6.4

The bit stuffing is used in CAN for two things. Firstly, it is used for bit synchronisation: as CAN is asynchronous (there is no separate clock signal) all nodes must remain synchronised by detecting rising and falling edges. Stuff bits ensure that there are always sufficient edges to maintain bit synchronisation.

Secondly, bit stuffing is used for error signalling. As it is an error for there to be 6 consecutive bits on the bus, then a node may deliberately send an *error frame* which starts with 6 dominant bits in order to signal that it has detected a fault. This will violate the bit stuffing rule on all nodes, so all nodes will send an error frame (whether they detected the original fault or not) and by the end of the error frame, all nodes in the system have abandoned the current frame. After the error frame all nodes resynchronise and may enter arbitration.

2.5.5 Undetected Errors

For communications, it is a theoretical impossibility to provide a completely reliable service [58] (unless certain assumptions are made in the fault model), and in any system there is a finite probability that an error will be undetected. In general a good communications system will be designed so that this probability is reduced to a level such that one has sufficient confidence that an undetected error will rarely occur.

In CAN, there are a number of error detection measures that when acting together reduce the probability to a very low level. It is useful to assess the *residual error*

[8] Only the start of frame, arbitration field, control field, data field and CRC sequence are stuffed.

probability of CAN—the probability that an application is presented with a data frame from the CAN bus which does not correspond exactly to a transmitted data frame.

Bosch, in the CAN specification, claim that the residual error probability of CAN is $4.7 \cdot 10^{-11} \cdot p_E$ [28], where p_E is the probability of a single bit on the bus changing value during the same interval. This is based on the CRC having a Hamming distance of 5 bits (is always able to detect up to 5 single bit changes). While it is true that the CAN CRC can guarantee to detect up to 5 single bit errors in the *data*, the CRC is actually applied to the un-stuffed bit stream, not the raw bit-stream that appears on the bus. This actually limits the error detection capability to having a Hamming distance of only 2 bits (in the worst case) [166].

An in-depth study by Charzinski [44] using more accurate modelling finds that due to this and other reasons, the residual error probability is much higher than first thought. It depends on p_E in a non-linear way. He states that the worst case residual error probability is $3.5 \cdot 10^{-9}$ and this occurs when $p_E \approx 0.02$.

Nevertheless, the residual error probability value is still very low. Therefore, the assumption as used throughout this thesis is justified, that *all errors caused by bus corruption are detected*.

2.5.6 Error Confinement Problems

Significant research on the role of CAN in high integrity systems has shown a variety of weaknesses of the protocol, but has also provided solutions to overcome these weaknesses.

Of particular concern is the fault confinement mechanism whereby nodes can turn themselves off (and on again) following high numbers of errors. The purpose of this fault containment mechanism is to prevent a *faulty* node from flooding the network with erroneous error frames or unnecessary retransmissions of data frames.

After modelling and analysis of this mechanism, Gaujal suggests that the bus-off state is reached too easily under bursts of interference [54, 55] and proposes a scheme to prevent this. On the other hand, a node failure has a high impact on the timing behaviour of the bus [141] so delaying bus-off is not without drawbacks.

An intermediate state, *error-passive*, where a node cannot signal faults has also been criticised because it can lead to inconsistent message delivery. A number of authors have suggested that the error-passive mode should be avoided [64, 50, 145]

2.5.7 Atomic Broadcast and Inconsistencies

The problem of atomic broadcast inconsistencies has been well documented [129, 144, 145, 126, 79]. Despite the claims of the CAN specification that CAN provides *atomic broadcast* (*i.e.* messages are delivered exactly once to all nodes, and in the same order that they were transmitted),[9] there are a number of scenarios in which CAN does not provide atomic broadcast.

Entering the error-passive state is one source of inconsistencies which is relatively easy to correct [145]. Other scenarios however, where the last-but-one bit of a data frame is received corrupted by a subset of nodes, leads to the possibility of double reception (where some nodes receive a frame twice and others once) or inconsistent message omissions (where some nodes receive the frame and others do not).

Studies have placed the probability of inconsistencies to be in the region of 10^{-6} failures/hour [145]. Solutions to the problem are varied. For many applications, it may not particularly matter if atomic broadcast is not always preserved. It is common practice to transmit only *absolute value* data, rather than increments or "toggle" messages so that a lost message does lead to a permanent erroneous state [170].

Alternatively, a number of higher level protocols have been designed to ensure consistent delivery for the various scenarios [145, 129, 98, 79] and with appropriate timing analysis [126]. Not all the protocols provide the same coverage of faults, and it should be noted that no protocol can ever provide a theoretically-perfect atomic broadcast property—instead a protocol aims to ensure that the scenarios which would lead to inconsistencies are so unlikely that the probability of failure is insignificant.

To summarise the description of CAN so far, evidence has been presented which indicates that CAN is a suitable protocol for systems with high reliability demands. A number of weaknesses are well-known. It is assumed for the rest of this thesis

[9] A more precise definition is given by Proenza [129] and others.

that either atomic broadcast inconsistencies are of no concern to the application, or that (where necessary) appropriate solutions are applied to prevent these weaknesses leading to failure. It is interesting to note that many other candidate protocols such as TTP, FlexRay, ARINC-629 and TTCAN cannot guarantee to provide consistent message delivery without higher level protocols, suggesting that atomic broadcast is not a common requirement.

2.5.8 Clock Synchronisation on CAN

Earlier, the clock synchronisation problem in a distributed system was introduced. There are a number of algorithms and protocols to synchronise clocks that are specific to CAN. They can be categorised into two broad groups: software level and hardware level. The chief difference is the resulting accuracy of the clocks. The symbol ε is used to denote the maximum deviation between any two clocks in the system.

Clock synchronisation at the software level can be run on any CAN controller, generally by running some distributed algorithm on all nodes. Software algorithms can ensure a maximum drift in the region of $\varepsilon = 100\,\mu$s. Rodrigues [137] provides a software based, fault tolerant clock synchronisation protocol specifically for CAN. The bandwidth overhead limits the precision of the synchronisation. By performing the synchronisation every 45 s, a precision of $\varepsilon = 100\,\mu$s can typically be obtained.

If special hardware support is available, such as may be found on some future CAN controller hardware to support TTCAN [61], or by design [165, 56] then a much finer clock synchronisation is possible. A hardware clock synchronisation mechanism may be used to ensure that the clocks do not drift by more than 1 bus bit-time ($\varepsilon = 1\,\mu$s).

2.6 CAN Worst Case Response Time Analysis

In Section 2.1.3, *deadlines* were described, and the concept that through the use of hard deadlines and some off-line analysis, the timeliness of a system could be *proven* with the same confidence as a functional property could be proven. This section shows how this may be achieved for communications on CAN.

To begin, considering the nature of a hard deadline: if all hard deadlines are met then the system is correct, otherwise it is not. A proof of timeliness shows that all hard deadlines are met. Logically therefore, the worst case scenario may be investigated, to show that deadlines are met in the worst case; if so then the deadlines must be met in all other cases. This is applied to CAN using *worst case response time analysis*. The approach is to find the scenario which can induce the worst case possible latency in the network and determine that latency.

Worst case analysis, and in particular worst case response time analysis (WCRT) are common techniques in real-time systems.

Analogy with CPU Scheduling

As Section 2.5.4 explained, CAN is a priority based protocol where collisions are avoided by using priorities for bus arbitration. CAN is therefore similar to a non-preemptive fixed priority CPU scheduling algorithm, and worst case response time analysis from the CPU scheduling domain [95] may be transferred almost directly to CAN.

2.6.1 Requirements for Worst Case Analysis

It is important before embarking on worst case analysis to determine the requirements for its validity. It will become clear later in this thesis that meeting these requirements is not as easy as implied in the following discussion.

The question that the analysis seeks to answer is: "if a node wishes to transmit a frame, what is the *longest time that may elapse* between the originating request and the frame being correctly received by another node". This bound is termed *worst case transmission latency* or *worst case response time* (WCRT).

Whether or not such a bound exists, (the *existence* of an answer) depends on several things, not least the amount and characteristics of other network traffic. Considering the CAN protocol, a requirement for bounded latency can be produced.

Lemma 1. *A bound on the transmission latency for a frame i exists if bounds exist on:*

the overhead of network activity associated with frame i and all network activity that takes preference on the bus over frame i.

The 'network activity' can be decomposed into a number of possibilities:

1. higher priority frames (which are always transmitted in preference);
2. any lower priority frame already begun transmission (which must complete or be lost due to faults);
3. error frames (which may interrupt any frame during transmission);
4. any other *inaccessibility* [167] due to faults, *e.g.* EMI, (which can interrupt any frame and cause an error frame).

There are also a number of other causes of delays which will not be considered directly, for example network partitioning or node crashes. Nevertheless, these cannot be totally ignored; temporary network partitioning for example can be caused by the CAN fault-containment mechanism which turns a node off after a large number of network faults. Rufino has considered many of these scenarios in more detail [140, 142].

From the list of network activities, it is clear that a bound exists on response time only if higher priority traffic is bounded and if the effect of faults is bounded [145, 157, 140].

2.6.2 Standard Periodic Stream Model of Traffic

The most frequently used model of network traffic in real-time systems is the *periodic stream* model where data traffic on the network consists of a number of *streams*, each stream being a sequence of frames where the frames are generated (queued for transmission) periodically with a known period. Typically, each stream would be transmitted using the same priority.

This model reflects common use of communication in distributed control systems. Control laws, notably PID control, are usually implemented as a periodic loop, therefore requiring sensor data, control data and actuator data to be transmitted periodically.

This model has the property that the overhead due to a periodic stream in a given interval is bounded. Hence, this model can be used to help provide a worst case response time.

The periodic stream model is used a number of times throughout this thesis; it is defined here for reference.

A stream i consists of a (possibly infinite) sequence of message frames. Each message frame has a maximum duration C_i when transmitted on the bus. Each message frame is created in response to a *trigger event*, which occurs periodically with a period T_i. Following a trigger event, the frame is queued for transmission on the bus according to the bus protocol.

Each stream may have a deadline, D_i, associated with it. In this thesis, as for most real-time literature, D_i is measured relative to the periodic trigger event.

Additionally, *jitter* may be introduced, where each trigger event may occur later than normal, by an amount of time no more than J_i. In practice, jitter may have many causes, in particular, scheduling algorithms on a CPU will usually not be able to provide an exact periodic interval for execution.

Also, *offsets*, O_i, may be applied to each stream, so that the release pattern of frames is controlled. In event-triggered systems, it is often appropriate to assume that offsets are unknown, or all offsets are 0.

A further use of the periodic stream model is that it is an upper limit to the *sporadic stream* model. The sporadic stream model is the same as the periodic stream model except that trigger events occur with a minimum inter-arrival time T_i, rather than exactly every period T_i. The worst case overhead of the sporadic model is when messages arrive every minimum inter-arrival time T_i, in which case it is equivalent to the periodic model.

Utilisation

Utilisation is a measure of how much of the bus capacity is used. Values range from 0 (nothing used) to 1 (fully utilised). For the periodic stream model, the utilisation of

each stream is easily calculated as

$$U_i = \frac{C_i}{T_i}$$

hence the total utilisation for the bus is derived [95]

$$U = \sum_{\forall i} \frac{C_i}{T_i} \tag{2.1}$$

Naturally, for the sporadic stream model, U also represents the 'worst case' or maximum utilisation possible.

2.6.3 The WCRT Analysis in Fault-free Conditions

In the absence of faults, worst case response time (WCRT) analysis of CAN for the periodic or sporadic models is straightforward [159]. It is performed in the same way as for non-preemptable processes on a CPU. The following equations for the worst case transmission time of each frame, R_i, can be derived [159].

Since faults are neglected, once a frame begins transmission, it completes. Hence,

$$R_i = w_i + C_i + J_i \tag{2.2}$$

where w_i is the *queueing delay* and corresponds to the maximum length of time that the frame may have to queue before beginning transmission. The worst case scenario which produces the maximum w_i can be derived from previous work on CPU scheduling [95] and corresponds to a *critical instant* when a lower priority frame has started transmission hence *blocking* a scheduling point and all higher priority streams generate a trigger event, causing maximal *interference*.

w_i is given by

$$w_i = B + \sum_{\forall j \in \mathsf{hp}(i)} \left\lceil \frac{w_i + J_j + \tau}{T_j} \right\rceil C_j \tag{2.3}$$

where B is the maximum possible time that a frame may have to wait for lower frames to complete, equal to the longest duration of lower priority frames; $\mathsf{hp}(i)$ is the set of

streams with a higher priority than i. The ceiling term derives the maximum number of trigger events that may occur in the interval w. The τ is necessary in non-preemptive analysis to eliminate the possibility of an optimistic result due to edge effects [18]. τ is the time of one bus clock tick. Hence the load due to higher priority frames (*interference*) is at most $\left\lceil \frac{w_i+J_j+\tau}{T_j} \right\rceil C_j$.

The smallest solution to equation (2.3) is the worst case queueing delay. However, the equation cannot be re-arranged to give a solution to w_i. Nevertheless the smallest w_i can be calculated numerically by forming a recurrence relation, as in equation (2.4).

$$w_i^{n+1} = B + \sum_{\forall j \in \mathsf{hp}(i)} \left\lceil \frac{w_i^n + J_j + \tau}{T_j} \right\rceil C_j \tag{2.4}$$

The solution is reached when $w_i^{n+1} = w_i^n$. As w_i^n is monotonically non-decreasing over n, iteration must start with a value of w_i^0 less than the solution, for example $w_i^0 = 0$.

For further details on this type of analysis, there are numerous good references dealing with its application in process scheduling [12, 11, 39].

2.6.4 Improving the WCRT Formulation

The equations in the preceding section are well known and have been used widely in academic literature [114, 167, 118] and industrial practice.[10] However there are a small number of changes which can usefully be made to the equations which simplify analysis later.

The differences between Tindell's original analysis [159] and the one used in the rest of this thesis are:

1. Separation of inter-frame space—a small correction is made concerning the inter-frame space that is transmitted between frames on the bus. Tindell considers that this is part of the frame, whereas CAN hardware is able to detect that a frame has completed immediately after the last bit of the frame. Therefore

[10] For example http://www.vector-cantech.com/.

Tindell's analysis is (very slightly) pessimistic as he includes the inter-frame space in the response times.

For further justification for this change, consider the lowest priority frame and the blocking it may receive. According to Tindell's original equations, this frame can receive no blocking, yet it may actually receive blocking time of 3τ because of the inter-frame space before it. Hence Tindell's blocking B is optimistic for the lowest priority frame (only). In practice however, this is not important because the optimism is exactly cancelled by the pessimism of including the inter-frame space at the end of the frame under analysis.

In future analysis in this thesis, the length of the inter-frame space, S, defined as $S = 3\tau$, is separate from the rest of the frame which is still given the symbol C_i.

2. Separation of Blocking and Interference parts—the blocking and interference parts of the worst case response time will be written as functions defined in separate equations.

3. Different form—a different form of the equation will be used which simplifies the mathematics for work later in the thesis. The essential difference is that instead of considering the queueing delay, the total response time is considered. The equations can be shown to be exactly the same by the substitution $t_i = w_i + C_i$.

The worst case response time equations which are used in the rest of this document are therefore defined as

$$R_i = J_i + t_i \tag{2.5}$$

and

$$t_i = B_i + C_i + I_i(t_i) \tag{2.6}$$

where:

C_i is the worst case transmission time [131] (the time it takes, in the worst case

to send frame i) assuming no errors, maximum bit stuffing[11] and not including the inter-frame space that follows the frame,

$$C_i = \left(44 + 8b + \left\lfloor \frac{34 + 8b - 1}{4} \right\rfloor\right) \tau \tag{2.7}$$

where τ is one bit-time and b is the number of data bytes in the frame (0 to 8). Due to bit-stuffing, the actual length of a frame depends on the data and arbitration value. For an 8-byte data frame, for example, the frame size can vary between 108 and 126 bits. C_i is understood to be the maximum possible length of a frame since this is a worst case analysis. A long-overdue study on the actual amount of bit stuffing that is typically observed provides a useful method to reduce its effects [118].

The worst case blocking, B_i is the maximum time a message may need to wait due to a lower priority message on the bus:

$$B_i = \max_{\forall k \in \mathsf{lp}(i)} (C_k) + S \tag{2.8}$$

where $\mathsf{lp}(i)$ is the set of messages with lower priority than i. Note that if i is the lowest priority frame then $B_i = S$.

Finally, $I_i(t)$ is the worst case interference that message i may receive in t time units:

$$I_i(t) = \sum_{j \in \mathsf{hp}(i)} \left\lceil \frac{t - C_i + J_j + \tau}{T_j} \right\rceil (C_j + S) \tag{2.9}$$

where $\mathsf{hp}(i)$ is the set of messages with higher priority than i and J_j is the worst case release jitter of frame j. Note that the numerator in the fraction involves $t - C_i$ as interference can only take place before frame i begins transmission (care must be taken never to require $I_i(a)$ if $a < C + J + \tau$). The τ is used to eliminate 'edge effects' in non-preemptive analysis where a high priority frame becomes ready as a medium priority one completes [18].

Equation (2.6) may be solved iteratively by forming a recurrence relation with $t_i^0 = C_i$ which terminates when $t_i^{n+1} = t_i^n$ or fails when $t_i^{n+1} > D_i - J_i$ where D_i is the

[11] A misinterpretation of the bit-stuffi ng rule led to an incorrect version of this calculation in some early literature [158] on CAN, where the denominator of the fraction was taken to be 5, not 4.

deadline and $D_i \leq T_i$.

If there is a solution $R_i \; \forall i$ and $R_i \leq D_i$ then the analysis above will guarantee that all messages will always meet their deadlines, provided that there are no faults.

2.7 Bus Faults on CAN

Section 2.6.1 listed the chief network activities whose effects must be bounded in order for worst case response time analysis to be possible. The periodic stream model was used to bound the effects of other frames on the bus, leaving only *inaccessibility* due to faults, retransmissions and error frames.

Chapter 1 suggested that the effects of EMI were difficult to measure and not possible to bound. Nevertheless, the use of a bounded fault model assumption is useful as it allows worst case analysis to be performed. A number of bounded fault models with significant merit may be considered. Some attempt to approximate real fault behaviour, perhaps derived from observation of real systems, others are simpler, aiming to create something of practical value by enabling simple worst case response time analysis to be conducted.

The means by which a bounded fault model is incorporated into worst case response time analysis is by adding an extra term to equation (2.6):

$$t_i = B_i + C_i + I_i(t_i) + E_i(t_i) \tag{2.10}$$

$E_i(t)$ is the worst case overhead due to network faults and extra frames that can occur in any given time interval, t. $E_i(t)$ must be bounded and non-decreasing over t. The equation can then be solved in the same manner as before.

2.7.1 Sporadic Fault Model

Tindell [159] suggested that faults could be regarded as sporadic single-bit faults. That is, there is a minimum separation between faults. Therefore faults can be treated in almost the same way as frames in the sporadic stream model. Additionally, to allow

for the possibility of faults occurring closer than the minimum separation, a single burst of faults is accepted (and the burst has a maximum length).

Each fault will cause an error frame to be transmitted and the retransmission of either a higher priority frame or if the fault falls in frame i then that frame will have to be retransmitted before it is received. In the worst case (*i.e.* where the fault falls on the last bit of a frame) then a fault leads to an overhead of the error frame and the retransmission (in addition to the the whole of the lost frame, which is already considered in the WCRT formulation).

This derives the following equation[12] for $E_i(t)$, the maximum overhead due to faults:

$$E_i(t) = \left(n_{burst} + \left\lceil \frac{t}{T_F} \right\rceil \right) \left(\max_{j \in \mathsf{hep}(i)} C_j + E \right) \tag{2.11}$$

where T_F is the minimum inter-arrival-time between faults and n_{burst} is the maximum number of faults that can occur in succession during a burst, E is the maximum length of an error frame (taken to be 29τ), $\mathsf{hep}(i)$ is the set of streams of higher or equal priority than i. The leftmost product-term of equation (2.11) is the maximum number of faults that can occur in an interval t, and the rightmost product-term is the maximum overhead of each fault.

The model is simple and effective but somewhat crude, there is no evidence that it reflects the nature of real faults. In use, it quickly leads to a very high overhead in the analysis which is not necessarily justified in practice. Nevertheless, in the absence of the ability to accurately measure or predict faults, this model is useful in that it gives a *guide* as to how much spare bandwidth is required to meet deadlines—a "fudge factor".

2.7.2 Sporadic Faults from Multiple Sources

Punnekkat [131] extends the sporadic fault model by providing a more general fault model which can deal with interference caused by several specific sporadic sources. The framework considers the most predominant forms of interference in a given appli-

[12] This equation is changed slightly in form from the original document [159] in order to present a consistent notation throughout this thesis.

cation (such as radar) and assumes that specific patterns of interference can be derived by monitoring a bus in proximity to the source. The patterns are characterised by an initial burst of faults and then a sporadic distribution of faults with a known minimum inter-arrival time. It is a form of bounded model and the overall effect can be described as a function $E_i(t)$ in the usual way.

The advantage of this approach is the ability to model specific interference patterns. However the disadvantages, like usual bounded fault model worst case analysis, are that it is difficult to have absolute confidence in the accuracy of the model and that it can easily result in a poor utilisation.

The formulation for $E_i(t)$ is summarised below. It consists of the summation of a number of different streams of faults:

$$E_i(t) = E_i^1(t) + E_i^2(t) + \ldots + E_i^k(t) \tag{2.12}$$

each one consisting of bursts and sporadic behaviour:

$$E_i^l(t) = Bu^l(t) \cdot (O_i^b + \max(0, I_b^l - \tau)) + Re^l(t) \cdot (O_i^r + \max(0, I_r^l - \tau)) \tag{2.13}$$

where $Bu^l(t)$ is the number of bursts in t and $Re^l(t)$ is the number of sporadic events in t. Separate overheads (O_i^b, O_i^r) can be defined for burst activity and sporadic activity. I_b^l and I_r^l represent the length of time that single faults affect the bus.

It is easily shown that the sporadic fault model is a special case of the multiple sources model with only one source and limited overheads. This model therefore is more general, allowing detailed expression of the overhead due to faults. However, the difficulties of providing such measurements with accuracy may prove to be prohibitive.

2.7.3 Bounded Omission Model

An alternative way to consider the fault model for CAN is to not model the faults directly, but instead to model at a higher level. The fault model for CAN established by Veríssimo and Rufino [140, 145, 143, 167] defines an *omission degree assumption*

that no more than *n* error and/or reactive overload frame transmissions are required to recover from errors in the transmission of an error frame, *i.e.* following a corrupt data frame, the model assumes that at most *n* retransmissions of that frame are required.

An analysis of CAN properties based on this assumption leads to a model which includes the following two properties [140] of interest in this context:

MCAN5—Bounded Omission Degree: in a known time interval, omission failures may occur in at most *k* transmissions.

MCAN6—Bounded Inaccessibility: in a known time interval, the network may be inaccessible at most *i* times, with a total duration of at most T_{ina}.

These properties (assumptions) form the basis of a range of work on CAN in high integrity systems [140].

2.7.4 On Bounded Fault Models for Analysis

The three frameworks described by Tindell [159], Punnekkat [131] and Rufino [140] illustrate ways to perform timing analysis of CAN under fault conditions. They all use models based on either a minimum inter-arrival time between faults or on a bounded omission degree. Therefore they assume that the overhead of faults that can occur is *bounded* and hence they fulfill the requirements for worst case response time analysis from Section 2.6.1

Previously,[13] it was suggested that because of the difficulties with predicting faults, such interference cannot be reliably characterised by a bounded model. This leads to two problems:

1. in order to have confidence that the analysis covers the worst case fault conditions, the worst case overhead due to faults must be 'set' very high (to guarantee all deadlines); this results in so much spare bandwidth being reserved that there is very little available for normal messages;

[13] §2.5.3 and Chapter 1

2. at run time, there is no guarantee that the faults will actually conform to the assumptions used for analysis.

Solutions to these problems are suggested in Chapters 3 and 4.

2.7.5 Non-bounded Fault Model

In an attempt to avoid the limitations of the bounded fault model, an alternative analysis, proposed by Navet [114] is based on a fault model with *random* fault arrivals. Faults are assumed to occur following a Poisson distribution, so in any given interval, there is a non-zero probability of any number of faults occurring—hence there is no bound.

The analysis provides an estimate of the *worst-case deadline failure probability* (WCDFP). This value is the probability that there are sufficient bus faults during the response time of a message to delay the message beyond its deadline. The concept may be then used to give an indication of the reliability of the system.

In more detail, the usual worst case response time analysis equations including worst case overheads due to errors [159] are used to calculate the smallest number of faults that can cause each frame to be delayed beyond its deadline. Then, based on a statistical fault model, they calculate the WCDFP as the probability that the number of faults would exceed the minimum number required to miss a deadline.

The approach is ‘worst-case’ because the underlying assumptions are that each frame always receives the maximum amount of interference possible and that bit errors occur at the end of each frame and hence cause the longest possible error recovery time. Yet, the use of a probabilistic fault model includes the possibility that there may be an unbounded number of faults in an interval.

The WCDFP should therefore be understood as “the probability of being in a situation where a deadline *may* be missed”. The conditional probability of missing a deadline (given that the bus has received such faults) is dependent on the task set and other factors. Here lies a source of pessimism in this approach which is explored in Chapter 4.

2.8 Scheduling on CAN

In its original (and most widely used) configuration, CAN provides non-preemptive fixed-priority scheduling. This section explores the implications of this and also a number of alternative scheduling policies that CAN is able to support.

2.8.1 Fixed-Priority Scheduling

CAN schedules messages using a fixed priority scheme: all frames have a fixed, known identifier which serves to both identify the frame to the receiver and to provide a priority for bus arbitration. Priority assignment can be arbitrary, but rate monotonic (RMPO) [95] or deadline monotonic (DMPO) [91] are useful and usually optimal. Although as Bate [18] shows, in a restricted synchronous system with only periodic tasks, DMPO is not necessarily optimal. However, for a CAN system, the restricted model does not usually apply and therefore it is normal to assume that DMPO is an optimum priority assignment for CAN.

It is generally acknowledged that fixed priority scheduling tends to have a lower utilisation than some *dynamic priority* scheduling schemes. The upper bound on utilisation for RM scheduling with guaranteed deadlines can be as low as 69% utilisation [95]. However, in practice, this figure is usually higher, for example, simulations by Lehoczky estimate that the average maximum utilisation for rate monotonic fixed-priority scheduling is around 88% for uniformly distributed processes [90].

There have been several alternative proposals for different scheduling policies on CAN in an attempt to either increase the bus utilisation, or improve the timing behaviour of CAN.

2.8.2 EDF on CAN

Earliest Deadline First (EDF) scheduling [95] is a well-known scheduling scheme for process scheduling, where at each scheduling point, the process with the closest deadline is selected for execution next. It has the advantage of potentially allowing a higher bus utilisation.

EDF on CAN has been attempted a number of times. One approach is to use the arbitration field to encode the deadline of each frame, so that the frame with the earliest deadline is always the next to be transmitted.

Livani and Kaiser [96, 78] suggest using the highest 8 bits of the longer CAN 2.0B 29-bit arbitration field to encode the deadline. The rest of the field is required for unique message identification and addressing. They further divide the priority field into three classes: hard, soft, and non real-time, which gives 127 possible values/slots for deadlines. This derives a *time horizon* of 8382 bit-times, which is 54 messages long. This scheme is then used to provide a four-tier scheduling scheme [97] where traffic with varying degrees of guaranteed deadlines and with non-guaranteed deadlines is scheduled in a loose TDMA style.

In order to avoid problems of a short time horizon and to avoid using CAN2.0B which has significantly more overhead due to the extended arbitration field, an alternative approach [172] chooses to schedule some messages by normal fixed priority scheduling and some by EDF scheduling. This creates a hierarchical bus scheduler capable of efficiently supporting a variety of traffic [171].

An extensive study into the problems and solutions of EDF scheduling on CAN is presented by Natale [112]. Although there is motivation for EDF scheduling on CAN, it is unclear that any benefits outweigh the difficulties. Changing the underlying scheduling policy is neither clean to implement nor free from overheads.

Centralised Control for Alternative Scheduling Schemes

Section 2.4.3 mentioned FTTCAN [4, 5, 2] which is a centralised control protocol implemented on CAN. The protocol is loosely TDMA based, but instead of the bus schedule being static, a master node periodically issues a broadcast message which contains information about which messages are to be scheduled and the ‘slots’ in which they are transmitted.

FTTCAN has also been used as the framework for implementing a number of centralised scheduling schemes on CAN, such as EDF and a constant bandwidth server [119].

2.8.3 TTCAN

A recent proposal is to make CAN into a time-triggered protocol. Deterministic timing behaviour has been seen as required [52] to support high integrity *x-by-wire systems*. TTCAN [75] is a higher level protocol that may be implemented on top of CAN to provide CAN-based TDMA communications. The goals of TTCAN are [52] to reduce response time variance ('jitter') and to impose a deterministic communication pattern on the bus. Prototype implementations of TTCAN exist [61].

The protocol is TDMA based, but with additional windows where certain messages may use normal CAN arbitration. Thus, TTCAN is able provide limited support for non-periodic information. Other merits of TTCAN are hard to determine as the protocol is not yet in wide use. Unlike TTP, TTCAN is implemented strictly on top of an event-triggered protocol, therefore the overheads of CAN (addressing *etc.*) are not avoided as they are in TTP, see Sections 2.5.4 and 2.4.1. The protocol is a work-in-progress item and likely to be superseded by FlexRay [19] before wide acceptance.

One of the most promising parts of TTCAN is the implementation of an accurate, high resolution clock on CAN through means of a periodic tick message [61] using hardware support.

2.9 Summary

This chapter has presented background material, including real-time theory, different forms of communication and a section regarding CAN and approaches to dealing with faults.

The concept of a hard deadline was introduced and how its use in conjunction with worst case response time analysis can make reasoning about the timing behaviour of a system manageable. Yet, this chapter has also described how using hard deadlines and worst case analysis to consider behaviour in the presence of faults has drawbacks, not least the fact that it can lead to an over-constrained and difficult to implement system. Practical ways of overcoming this are the topics of the next two chapters.

3 Weakly-hard Analysis of CAN

HARD DEADLINES PROVIDE A RIGOROUS FOUNDATION for the design of real-time systems. However, as Chapter 2 noted, *constructing* systems to guarantee hard deadlines has many difficulties. In particular, in a noisy environment where faults on the bus are either very frequent or difficult to determine, guaranteeing hard deadlines on communication can become impossible. Many real-time applications, however, do not *need* all communication deadlines to be guaranteed—because the applications have some robustness (either naturally, or by engineering) to timing faults. In this Chapter, a softer, more flexible, form of deadline, called a *weakly-hard deadline* [22] is introduced into the world of real-time communication as a practical way of engineering such systems.

The area of *weakly-hard real-time systems* [21] is a growing area of work, studied originally by Bernat. It concerns the specification and analysis of real-time systems where a *specified* number of deadlines may be missed. In its original context, weakly-hard analysis was used to analyse processes on a CPU. The contribution in this chapter of this thesis is to apply weakly-hard analysis to the area of communication, specifically to CAN, in order to provide useful guarantees of performance in the presence of high or unknown levels of faults.

3.1 Motivation

The motivation for a weakly-hard system is that in many practical engineering contexts, 'occasional' missed deadlines can be tolerated, recall from Section 2.4.2 the *firm deadline* model used by avionics applications using ARINC-629. However, it is still important to be able to guarantee that *sufficient* messages are delivered on time

in order for the system to maintain performance or stability. By considering exactly which deadlines are actually required to be met, rather than simply classifying everything as a 'hard' deadline, a bus may be able to provide guarantees yet run at a higher utilisation and tolerate more faults.

By examining a relevant application that uses a real-time bus: a distributed PID[1] controller, further evidence can be found that in practice it is not necessary to meet all deadlines (*i.e.* to describe a deadline as 'hard' is over-constraining). Control laws are generally able to work effectively if a small number of data values are lost; significant research has been done on how to maintain stability when messages are omitted (vacant sampling) [116], or with sampling jitter [104]. For occasionally delayed output data, this may be incorporated into the parameters of the control law. Further justification for *missing some deadlines in a predictable way* is given by Bernat [21].

Therefore, in order to avoid placing extra requirements on the communication system to guarantee that *all* messages are timely (when in fact if some are late then the software can easily cope) notions are considered that are softer than hard deadlines. In particular, weakly-hard deadlines allow provide a framework to specify exactly how many deadlines may be missed in the worst case. For example: "*in any 20 consecutive deadlines the process must always meet at least 18 of them and it must never miss any 2 consecutively*". Weakly-hard response time analysis may then be used to show that all the weakly-hard constraints are met.

The rest of this Chapter explains the intuition behind weakly-hard analysis and how it may be exploited on CAN. Section 3.3 reviews some weakly-hard theory from previous work. This is used in Section 3.4 where the details of the analysis of CAN are explained. Weakly-hard constraints on CAN are evaluated in Section 3.9 by means of a case study. This leads to an observation about designing the system to make best use of a flexible bus by using non-harmonic periods. Finally, the limitations of the bounded fault model are addressed and weakly-hard constraints are evaluated against a different fault model.

[1] PID—Proportional, Integral, Derivative; a standard way of implementing feedback control.

3.2 Intuition

Before considering weakly-hard deadlines in detail, it is useful to leap ahead for a little insight into the workings of a fixed priority event-triggered system. The intuition behind weakly-hard analysis is to exploit the fact that in an event-triggered system, tasks (or message frames) tend to suffer different levels of interference on each invocation. In a periodic stream model, the interference for each invocation depends on the relative periods of higher priority tasks (frames). It can be seen by measurement, simulation and analysis that there is huge variation in the response times for each invocation, especially where the periods of higher priority streams are non-harmonic. In general, the average case response time is *much* lower than the worst case response time.

As an example of this variation, Figure 3.1 shows the distribution of worst case response times for some consecutive invocations of a frame on a CAN bus from the analysis later in this chapter. The frame is priority 2 (low) of the message set described later in Table 3.2, assuming no faults. The worst case response time is 23.016 ms but as can be clearly seen, the response time at other invocations within the hyperperiod is much shorter.

The important point to note about the graph is that there is only a relatively small number of peaks: most of the *mass* of the graph is well within the worst case.

Next consider exactly the same graph, but with a possibility of one fault occurring within the hyperperiod (Figure 3.2). All the bars have increased slightly, indeed two of them have exceeded the deadline. However, the other 248 invocations are well within the deadline and are guaranteed to be on time.

Weakly-hard analysis can be used to exploit this variation in response times in a flexible bus to provide deadline guarantees even though the worst case may not be guaranteed.

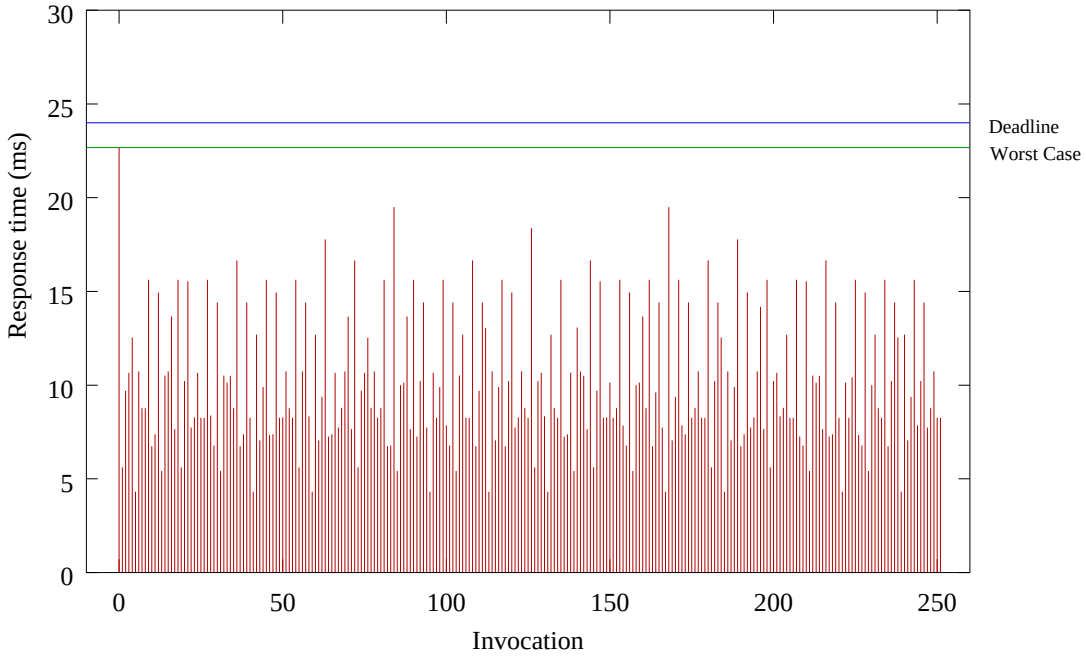


Figure 3.1: Typical Distribution of Response Times.

3.3 Weakly-hard Theory

As the previous section indicated, there are many practical engineering contexts where it is not necessary to meet *all* deadlines: very few 'hard' real-time systems (subsystems) are actually 'hard' in the sense of *needing* to meet all deadlines. However, neither are they systems with no real-time requirements; missing too many deadlines does matter. Nearly all control laws fit this scenario where occasional deadline misses of input/output data are easily tolerated, but minimum levels of timely behaviour are necessary in order for the system to function correctly or efficiently.

Such systems are termed 'weakly-hard' real-time systems [21], and they can be characterised by considering the numbers of deadlines that are missed and met. In order to argue about missing deadlines, a formal notation is required to specify the number of missed messages.

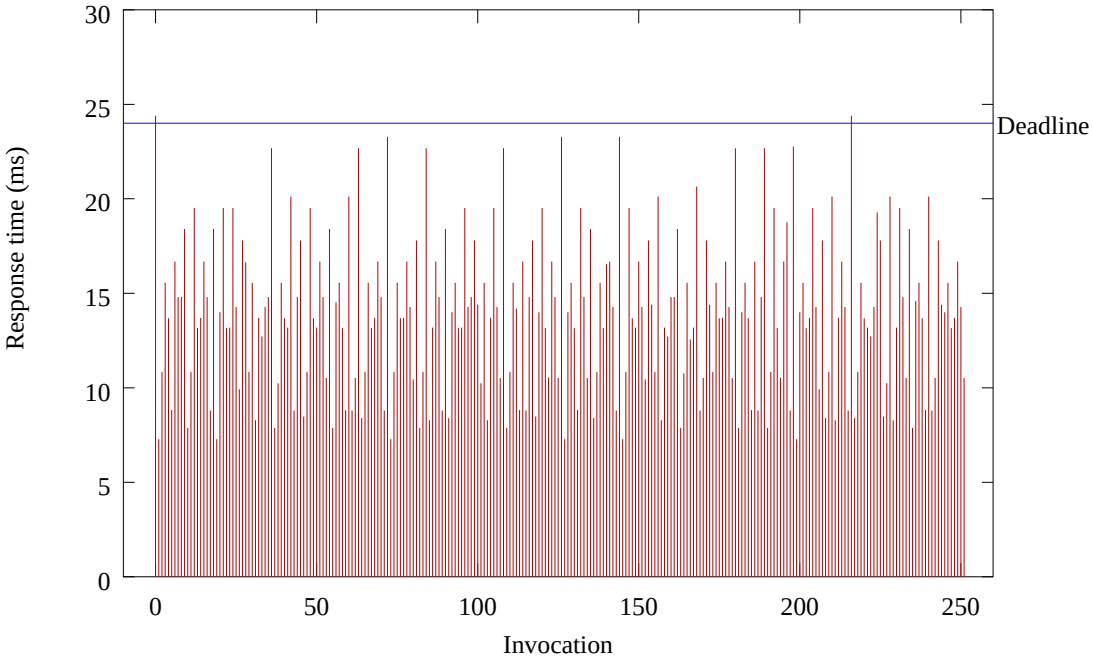


Figure 3.2: Typical Distribution of Response Times with One Fault.

3.3.1 Weakly-hard Notation

Four basic notational constructs, 'constraints', are defined by Bernat [23] which shall be used in this document:

- $\binom{n}{m}$ for any m consecutive deadlines in a stream, at least n are met (messages arrive and arrive on time). No constraint is put on whether these n messages arrive consecutively or non-consecutively.
- $\overline{\binom{n}{m}}$ for any m consecutive deadlines in a stream, at most n deadlines can be missed. This is the dual of the previous constraint. $\binom{n}{m}$ is equivalent to $\overline{\binom{m-n}{m}}$.
- $\left\langle \begin{smallmatrix} n \\ m \end{smallmatrix} \right\rangle$ for any m consecutive deadlines in a stream, at least n *consecutive* deadlines are met. This is clearly more demanding than just meeting n in a window.
- $\overline{\langle n \rangle}$ it is not the case that n consecutive message deadlines are missed in a row. In other words, $\overline{\langle 3 \rangle}$ means that 3 deadlines in a row are not missed.

Weakly-hard notation can be used in a number of ways. There are three uses that are particularly of merit:

Specification. Weakly-hard notation can be used to specify *requirements*, *e.g.* "The application requires $\binom{17}{20}$ (that 17 in 20 messages arrive on time)."

Guarantee. A weakly-hard *guarantee* specifies an upper bound on the number of missed deadlines during a *future* window of time as determined by analysis, *e.g.* "The bus guarantees $\binom{18}{20}$ (that 18 in 20 deadlines will be met)."

Observation. An *observed* weakly-hard behaviour is derived from a long simulation or from monitoring of the system, *e.g.* "The bus actually achieved $\binom{19}{20}$ (always met 19 in any 20 consecutive deadlines)."

All of these are termed weakly-hard constraints. Further, it is useful to measure how often a given constraint is achieved or not achieved, for example "*The bus achieved* $\binom{20}{20}$ *99.46% of the time*".

This concludes the brief review of weakly-hard deadlines. The following sections present the application of weakly-hard analysis to CAN, which is the main contribution of this Chapter.

3.4 Guaranteed Weakly-hard Analysis of CAN

In this section a weakly-hard analysis of CAN including faults is presented. The overall approach of weakly-hard analysis is to calculate the worst case response time, $R_{i,k}$, for each possible invocation, $k = 0, 1, 2, \ldots$, of each frame i (within a hyperperiod). The CPU scheduling approach [23] is modified from the preemptive process model to the non-preemptive CAN scheduling model with preemptive faults. The worst case response times are then mapped to weakly-hard constraints which form guarantees of future behaviour.

3.4.1 Terminology

It is first necessary to introduce some scheduling terminology [21] that will be used in the subsequent analysis.

The periodic stream model (see Section 2.6.2) is used—all messages (within each stream) are released periodically, with known periods. Perfect clock synchronisation is also assumed, although this is relaxed later in Section 3.8.

Therefore, it can be stated that the *pattern of release times* (trigger events) of *all* messages is repeated every *hyperperiod* which is $H = \text{LCM}\{\forall i : T_i\}$ time units long [21].

A message at priority i will only suffer interference from higher priority messages. Therefore the pattern of interference and therefore the pattern of worst case response times repeats every *hyperperiod at level i*, given by $H_i = \text{LCM}\{\forall i \in \mathsf{hep}(i) : T_i\}$, where $\mathsf{hep}(i)$ is the set of messages of higher or equal priority to i.

The exception to this is that there may be an overrun at the end of the first hyperperiod into the second hyperperiod (and likewise into the third etc). Therefore response times in the second hyperperiod may be longer than those in the first hyperperiod. However, as Bernat shows [22] there is no need to consider further hyperperiods as the response times in successive hyperperiods cannot exceed those of the second, provided that the utilisation $U \leq 1$.

The number of invocations of a message in a hyperperiod is given by $A_i = H/T_i$, and the number of invocations of a message in the hyperperiod at level i is $a_i = H_i/T_i$.

3.4.2 Calculating Response Time $R_{i,k}$

The worst case response time of the k^{th} invocation within the hyperperiod of stream i is $R_{i,k}$. It may calculated using an extension of the worst-case response time formulation technique for multiple invocations [22] which is updated to include interference from CAN error frames and consider the non-preemptive nature of CAN data frames.

For the following calculations, it will be assumed that at time $t = 0$, the first frames in all message streams are released. Therefore, the release time of each frame is defined as

$$S_{i,k} = kT_i \quad k = 0, 1, \ldots a_i - 1 \tag{3.1}$$

With reference to Figure 3.3, the worst case response time, $w_{i,k}$, *relative to the*

critical instant is the sum of the times taken by all events on the bus up to $w_{i,k}$:

$$w_{i,k} = k(C_i + S) + C_i + (k+1)B_i + I_i(w_{i,k}) + E_{i,k}(w_{i,k}) + \delta_{i,k} \tag{3.2}$$

That is: previous and current invocations of message i, blocking and interference (calculated in the usual ways, see §2.6.4 on page 65) and the worst case error overhead. $\delta_{i,k}$ is the amount of *idle time at level i* between $t = 0$ and $S_{i,k}$, the 'spare' bus capacity (see below).

The value $w_{i,k}$ is calculated iteratively in the familiar manner using a recurrence relation, as described in Section 2.5, starting with a small value, such as C_i. From $w_{i,k}$, the actual worst case response time relative to the planned release at $S_{i,k}$ can be calculated:

$$R_{i,k} = w_{i,k} + J_i - S_{i,k} \tag{3.3}$$

3.4.3 Calculating Idle Time $\delta_{i,k}$

Equation (3.2) made use of $\delta_{i,k}$ which is the maximum length of time that the bus is available for use by lower priority frames in the interval $(0, S_{i,k}]$ (excluding time that has already been accounted for in equation (3.2), so the blocking factor B_i is not included in the calculation). $\delta_{i,k}$ is termed the *idle time at level i*.

$\delta_{i,k}$ can be computed by considering the bus behaviour after introducing a single *virtual preemptable message* at a priority level just below i that consumes all unused bandwidth between $t = 0$ and the release of the frame at $S_{i,k}$. Let $\bar{\tau}$ be the virtual message. The idle time, $\delta_{i,k}$, is the largest $\bar{C_\tau}$ that makes the virtual message schedulable (see Figure 3.3):

$$\delta_{i,k} = \max\{\bar{C_\tau} | \bar{R_\tau} \leq S_{i,k}\} \tag{3.4}$$

where $\bar{R_\tau} = u$ is the solution to the following equation:

$$u = k(C_i + S) + kB_i + \bar{C_\tau} + \sum_{j \in \mathsf{hp}(i)} \left\lceil \frac{u + J_j}{T_j} \right\rceil (C_j + S) + E_{i,k}(u) \tag{3.5}$$

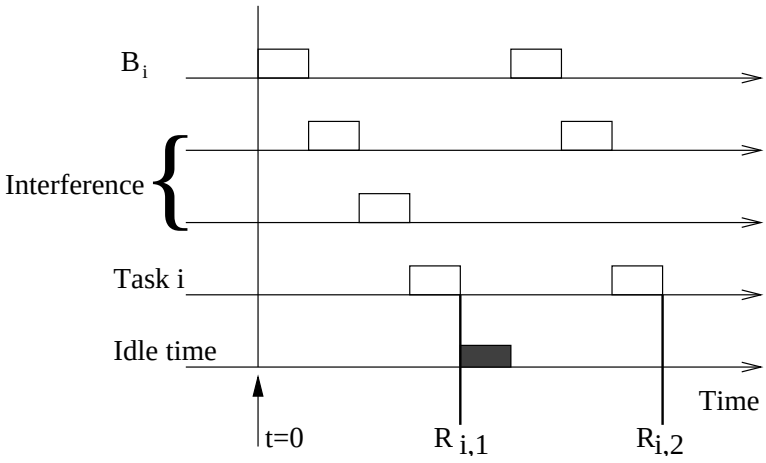


Figure 3.3: Virtual Task to Calculate the Idle Time.

Equation (3.5) is very similar to equation (3.2), the major exception being the interference term which is required to be preemptive, rather than the non-preemptive form of equation (2.9). Other than this, the equation must match the main response time equation (3.2) exactly so that between $t = 0$ and $t = S_{i,k}$, the worst case schedule that the idle time calculation is based on is exactly the same as the schedule that the response time equation is based on.[2]

Note that $\delta_{i,k}$ in equation (3.2) does not depend on $w_{i,k}$, therefore it does not need to be reevaluated at every iteration.

3.5 Incorporating Faults: Calculating $E_{i,k}(t)$

The multiple invocation response time analysis in the previous section included an error term $E_{i,k}(t)$ in the same way as the CAN worst case response time analysis did from Section 2.6.4. This is the bounded overhead due to faults in any interval of length t.

Any bounded fault model could equally be used for the analysis, depending on the application. Here the sporadic fault model [159] explained in Section 2.7 will be used,

[2] To aid understanding of this formulation, note that $\delta_{i,k}$ is not the same as u. The relationship between $\delta_{i,k}$ and u was defined in equation (3.4).

that considers single-bit errors that have a minimum inter-arrival time, T_F (the burst error term n_{burst} from equation (2.11) is omitted here). Therefore an expression for the worst case overhead due to faults during an interval t is:

$$E_i(t) = \left\lceil \frac{t}{T_F} \right\rceil \max_{k \in \mathsf{hep}(i)} (C_k + E + S) \tag{3.6}$$

where E, is the longest possible error frame resulting from a single fault.

However, incorporating the error function is not as straightforward as might be expected due to the nature of the worst case(s). When considering only one invocation, the worst case fault scenario is easy to determine: a fault occurs at the critical instant, then every T_F afterwards.[3]

However, although the *periodic stream model* was assumed for messages, faults are assumed *sporadic*. Therefore the arguments about release patterns do not apply to faults. Instead, each invocation of a frame has an individual worst case scenario for faults which must be determined. The worst case scenario is where a fault occurs at $S_{i,k}$ then faults re-occur at their fastest rate after $S_{i,k}$. In other words, there is a critical instant *for faults* at every invocation.

Therefore, at each invocation k, a new worst case scenario is analysed. In order to ensure that idle calculations are correct, for each invocation the sporadic fault model is mapped to a periodic fault model but with an offset O_F such that a fault occurs at $S_{i,k}$ for the invocation under analysis, and faults occur periodically before and after $S_{i,k}$:

$$O_F = S_{i,k} - \left\lfloor \frac{S_{i,k}}{T_F} \right\rfloor T_F \tag{3.7}$$

The overhead due to this is then calculated by using a variation of offset based analysis:

$$E_{i,k}(t) = \left\lceil \frac{t - O_F}{T_F} \right\rceil \left(\max_{k \in \mathsf{hep}(i)} C_k + E + S \right) \tag{3.8}$$

This concludes the weakly-hard response time analysis. The following sections show how the analysis may be applied to good effect on an event-triggered bus.

[3] Note that this is not actually the worst case scenario that may occur on CAN, but *within the fault model* (that each fault causes the maximum possible overhead no matter where in a frame it occurs) then this scenario is equivalent to the worst possible real case on CAN.

3.6 Conversion to Weakly Hard Constraints

Once the worst case response times, $R_{i,k}$, for each invocation have been calculated, it is a simple matter to convert these to weakly-hard constraints, by scanning the response times and comparing $R_{i,k} \leq D_i$. Checking the satisfiability of system behaviour of length p against a weakly-hard constraint can be done with simple algorithms of cost $O(p)$. For further details on properties of weakly-hard constraints and algorithms (including the use of μ-patterns as a useful intermediate format) see previous publications on weakly-hard systems [23, 25, 21].

3.7 Critical Instants and Harmonic Periods

Early in this chapter, Figure 3.1 on page 78 was used to show that different invocations of a frame may have different worst case response times and hence motivate the idea that weakly-hard analysis can be used to provide a more useful guarantee than just hard deadline WCRT analysis. This section discusses the implications that the critical instant and periods have on the variation of the worst case response times and the length of the hyperperiod.

Considering the invocations of frame i: if the period of stream i is a multiple of the period of a higher priority stream j, ($T_i = nT_j, n \in \{1,2,\ldots\}$) then (in the worst case) every time i is released then j is also released and so j will always provide maximum interference for i. Applying this to all higher priority messages, where T_i is a multiple of all higher priority periods, then all higher priority messages will always interfere with i. Therefore $R_{i,k} = R_{i,0}, k \in \{1,2,3,\ldots\}$, so message i gains nothing from weakly-hard analysis.

This means that every invocation of i is the worst case—the effect seen in Figure 3.1 does not occur, instead the scenario in Figure 3.4 is seen. The diagram shows a simplified (blocking is ignored) schedule with two harmonic periodic streams. It is possible for every invocation to receive the same interference as the worst case.

On the other hand, if T_i is not a multiple of T_j then the potential for j to cause maximum interference to i is reduced (the amount of interference is calculated using

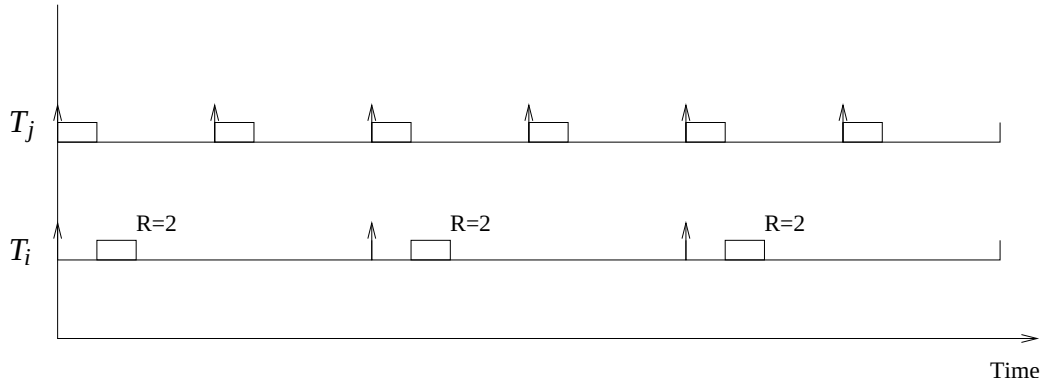


Figure 3.4: Harmonic Periods: $T_j = 4, T_i = 8$.

the equations in the previous section). Consider Figure 3.5 where even though the higher priority stream has a shorter period ($T_j = 3$) and therefore the bus loading is higher, for every two in three invocations, the worst case scenario ($R_i = 2$) cannot be observed.

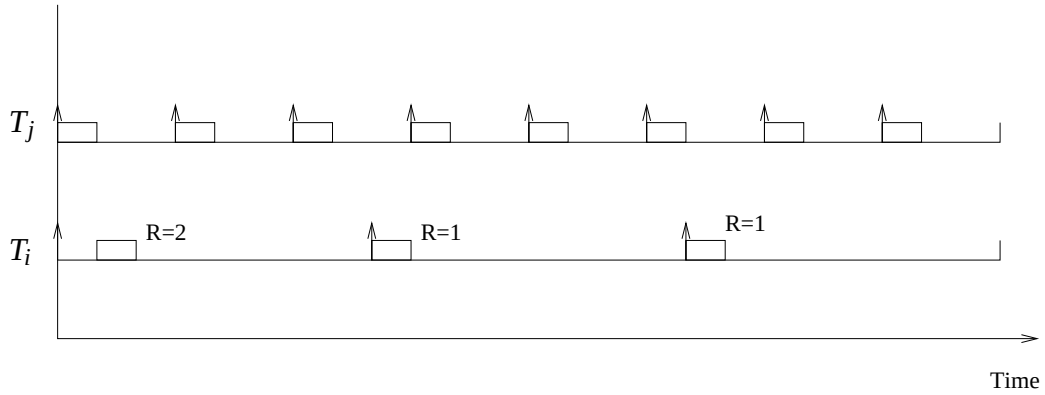


Figure 3.5: Non-harmonic Periods: $T_j = 3, T_i = 8$.

It is clear that non-harmonic periods cause variation of the set of frames which may cause interference to the different invocations in a lower priority stream. Hence non-harmonic periods are necessary for variation in the *worst case* response times.

Definition 1. A periodic stream *i* is *non-harmonic* if its period is not a multiple of any higher priority period.

$$\forall j \in \mathsf{hp}(i), \nexists n \in \mathbb{N} | T_i = nT_j \tag{3.9}$$

Definition 2. A *non-harmonic message set* is a message set where no period is a multiple of the period of any higher priority message. That is:

$$\forall i, \forall j \in \mathsf{hp}(i), \nexists n \in \mathbb{N} | T_i = nT_j \tag{3.10}$$

For a non-harmonic stream i, since no higher priority period is a multiple of T_i, a higher priority frame can only be released at the same time as a frame from stream i once per hyperperiod at level-i. This does not imply that the worst case interference occurs only once every hyperperiod at level-i since there are usually several scenarios which are equivalent to a critical instant. However the effect of a non-harmonic message-set is to extend the hyperperiod (and all i-hyperperiods) thereby reducing the *frequency* of situations where a number of frames are released simultaneously. Non-harmonic message sets are likely to gain more from weakly-hard analysis than sets with harmonic periods.

However, a system where the periods are strictly non-harmonic can be constraining to design and difficult to analyse because the extended hyperperiod quickly becomes extremely long, requiring billions of calculations to determine the response times. It has been observed that in practice, a strictly non-harmonic message set is not needed; a message set that exhibits some non-harmony is sufficient to benefit from weakly-hard analysis.

Techniques for determining non-harmonic periods and deciding how long a hyperperiod needs to be are outside the scope of this thesis and is considered to be an open research area. In general, the presence of sufficient non-harmony may be detected by comparing the results of weakly hard analysis to a desired weakly-hard specification. Typically (when the streams are ordered rate monotonically, RMPO) if there are more than about 200 invocations in a hyperperiod at level i ($a_i > 200$) then useful gains can be made. Such an example appears later in Section 3.9.3.

Introducing non-harmony in a system by changing periods may seem at odds with traditional design where harmonic periods tend to be used. However, one reason that harmonic periods are often used is to allow easier integration into a static time-triggered schedule. Yet with a flexible, event-triggered bus and similarly flexible priority based scheduling on the nodes, such a requirement is superfluous. Instead, periods

can be chosen without restriction in order to choose the optimum value *for the application*. Such values are unlikely to be harmonic, unless of course 'round number' periods such as 100 ms are used arbitrarily in the absence of any other requirements. Unfortunately, this is frequently observed to occur in practice [17].

A general method for determining periods is to specify periods not as single values, but as either a set of possible values [106], or a range of acceptable values [17] by exploiting the ability of applications to work within bounds [16]. This would give some flexibility for a search algorithm to manipulate precise periods to determine a suitable length hyperperiod.

The whole issue of choices when designing timing requirements is an open research area [17, 106].

3.8 Relaxing Clock Synchronisation

Previous weakly-hard theory [21] was for processes on a single node, were it is reasonable to assume that all processes may share a single clock. The consequence of this is that the pattern of release times is repeated *exactly* each hyperperiod. However, with CAN, where frames are sent from different nodes, there is no such single clock. Each node must have its own clock and these clocks will certainly drift relative to each other.

Initially for the analysis in Section 3.4, perfect clock synchronisation was assumed. In this section, the restriction is relaxed by assuming that a clock synchronisation algorithm is used to ensure that all clocks in the system read approximately the same time. As discussed in Section 2.5.8, there are numerous clock synchronisation protocols which can ensure that clocks are approximately synchronised, within a known bound, including some particular examples that are well suited to CAN [165, 56, 137]. These algorithms all are able to ensure that the difference between any pair of clocks is no more than some value ε.

Lemma 2. *The effects of Bounded Clock Difference can be incorporated into the response time analysis by considering the maximum drift, ε, to be an additional form*

of jitter in interference calculations:

$$I_i(t) = \sum_{j \in \mathsf{hp}(i)} \left\lceil \frac{t - C_i + J_j + \varepsilon + \tau}{T_j} \right\rceil (C_j + S) \tag{3.11}$$

Proof. Since clocks remain approximately synchronised (rather than drifting indefinitely), the overall structure of the hyperperiod remains intact. Nevertheless, there may be some variation of the release times due to the clock drift. The worst case effect of which is to allow any higher priority frame to cause early interference to any other frame by at most ε (as viewed from the perspective of the lower priority frame), which is exactly equivalent to the effect of jitter. □

However, in practice this effect will be so small that other pessimism introduced by the analysis (such as from bit stuffing, blocking *etc.*) will be many times worse than the drift effect. Unless ε is large, adding it to the equation will not usually change the results of the analysis.

3.9 Evaluating Weakly Hard

The motivation for using weakly-hard constraints was to take advantage of the flexibility of event-triggered communication for fault-tolerance. The aim of this section is to evaluate the benefits of using weakly-hard constraints on CAN. In order to do this, example CAN based systems will be considered at a variety of fault levels. The evaluation considers the ability of weakly-hard analysis to describe and predict system behaviour with high levels of faults and the ability to describe behaviour in sufficient detail that a system designer may gain fuller understanding of the behaviour.

Two suitable message sets are chosen, one is taken directly from a well-known benchmark, the second is a partially non-harmonic modification of this to show how to best take advantage of weakly-hard analysis. Then, in order to consider the ability of a weakly-hard system to tolerate faults, the following are done in logical progression:

- Normal worst case response time analysis is performed at various fault levels. At high levels of faults, the analysis indicates that no guarantees can be given.

- Multiple invocation worst case response time analysis was performed at the same levels of faults and the results mapped to weakly-hard constraints. This provides guarantees for a minimum number of frames at much higher fault levels.

- Simulation of CAN bus is compared to the weakly-hard worst case response time analysis guarantees to provide an indication of the pessimism in the weakly-hard analysis.

3.9.1 SAE Benchmark Message Set

The Society of Automotive Engineers (SAE) produced a set of messages [147] that might be transmitted within a vehicle. This message set has frequently been used as a benchmark to evaluate CAN and has been the source of some discussion [158, 81].

The original benchmark contains 53 messages transmitted from 7 nodes. It is unschedulable at the lowest data-rate (125 kbit/s) on CAN: the worst case[4] (hypothetical) bus utilisation is 125.29% so the bus could lag further and further behind. Certainly no deadline guarantees can be made. At higher data-rates, the benchmark is easily schedulable, but the lowest rate is preferred because of the fault-tolerance advantages in the physical layer at this speed, see Section 2.5.3.

The SEA benchmark contained many 1-byte data messages; this results in a very low efficiency (because the ratio of useful data to header information was low). Tindell produced an equivalent benchmark [158] by 'piggy-backing' periodic messages together that originated from the same node, and then combining sporadic messages together by using periodic 'server messages'. This resulted in a set of 17 periodic messages that are schedulable at the slowest data-rate with a worst case utilisation of approximately 85%.

[4] Assuming maximum bit stuffing. That is, the actual data transmitted causes the maximum amount of additional 'stuff bits' to be inserted into the stream, see Section 2.5.4 and equation (2.7) in Section 2.6.4.

The Tindell benchmark is in Table 3.1, with the exception that each message has been given a jitter component of 200μs. This is a small, but realistic value, used by Tindell [158] when considering worst case response time analysis with jitter. This value shall be applied for the message set used in this thesis. For comparison, the table includes the worst case response times with no faults. This data-set is only just schedulable; the high utilisation means that there is very little bandwidth available for recovery from faults. Note that several messages (12, 9, 8) have a worst case response time of just below their deadline: even one fault can cause these messages to miss their deadlines.

Therefore, although this benchmark may not match a real system (more spare bandwidth to tolerate faults might be expected), the benchmark provides an ideal basis for creating challenging tests for fault tolerance. This benchmark is used and weakly-hard analysis is applied to it.

Table 3.1: Tindell's SAE Benchmark, as used in this thesis.

i	Bytes	C_i (ms)	T_i (ms)	J_i (ms)	D_i (ms)	R_i (ms)
17	62	0.496	1000	0.2	5	1.616
16	72	0.576	5	0.2	5	2.216
15	62	0.496	5	0.2	5	2.736
14	72	0.576	5	0.2	5	3.336
13	62	0.496	5	0.2	5	3.856
12	72	0.576	5	0.2	5	4.456
11	112	0.896	10	0.2	10	5.216
10	62	0.496	10	0.2	10	8.576
9	72	0.576	10	0.2	10	9.176
8	72	0.576	10	0.2	10	9.776
7	62	0.496	100	0.2	20	10.296
6	92	0.736	100	0.2	100	19.296
5	62	0.496	100	0.2	100	19.816
4	62	0.496	100	0.2	100	20.336
3	82	0.656	1000	0.2	1000	29.176
2	62	0.496	1000	0.2	1000	29.696
1	62	0.496	1000	0.2	1000	29.72

3.9.2 Weakly Hard Analysis of SAE Benchmark

As an example to evaluate the weakly-hard analysis; stream 8 will be considered. There are $A_8 = 100$ invocations of frame 8 in the hyperperiod $H = 1000\,\text{ms}$.

Consider the effect of exactly one fault within the hyperperiod, $T_F = 1000\,\text{ms}$. The overall worst case response time of this message, equation (2.5), with one fault is greater than the deadline ($R_8 = 18.648\,\text{ms}$); this message would be considered unschedulable (since a *hard* deadline cannot be guaranteed).

However, considering all invocations in the hyperperiod by applying equation (3.3) it can be seen that this response time can occur only once within the hyperperiod, at the critical instant. The results of the weakly-hard response time analysis are:

$$R_{8,k} = \begin{cases} 18.648\,\text{ms} & k = 0 \\ 10.008\,\text{ms} & k = 1 \\ 9.088\,\text{ms} & k = 2 \\ 9.056\,\text{ms} & k \in 3..99 \end{cases}$$

This may be mapped to weakly-hard notation to give the following guarantees:

$$\binom{98}{100}, \overline{\binom{2}{100}}, \left\langle \begin{matrix} 49 \\ 100 \end{matrix} \right\rangle, \overline{\langle 3 \rangle}$$

This means that if there is at most one fault within the hyperperiod then the message is *guaranteed* to arrive before the deadline (10 ms) at least 98 times out of 100. Only if the fault happens to occur near to the critical instant is it possible to have this message delayed beyond the deadline. It is not possible to miss three deadlines in a row. Notice that the worst case response time for 98% of the invocations is less than half the worst case response time.

Of the worst case response times for this message, 97 of them are the same. This is because all the higher priority messages except the highest priority have periods that are harmonic with the period of this message; therefore at all but the *one* invocation of the message in the hyperperiod at level-8, there is exactly the same potential for interference. The first invocation is different because it is the only message able

to suffer interference from the highest priority message. The second and third invocations are different because the first and second invocations finish after the next is queued ($R_{8,0} > S_{8,1}$), therefore the next invocation also suffers 'interference' from the previous invocation of the same message.

Similar results are evident for the other messages in the set. Weakly hard constraints have been evaluated, and for this message set, the improvement over hard worst case analysis is small, but nonetheless useful.

3.9.3 Partially-nonharmonic SAE Benchmark

As indicated in Section 3.9.1, weakly-hard analysis does provide merit compared to normal response time analysis. However, the SAE benchmark and Tindell's compressed message set have messages with harmonic periods, perhaps in order to facilitate cyclic scheduling. In an event-triggered system, there is no need to require the periods to be so constrained, indeed, as discussed in Section 3.7 when using weakly-hard constraints, it is better if they are not harmonic. This is illustrated in the following analysis of a partially non-harmonic variation of the previous benchmark.

A new message set appears in Table 3.2, based on the SAE benchmark, but with the periods adjusted to remove some harmony. The new message set is not completely non-harmonic (as this would lead to an extremely long hyperperiod which would be expensive to compute), but the messages do exhibit a lot of non-harmony. The hyperperiod is 252 s and it provides a maximum[5] bus utilisation of about 73% at 125 kbit/s. The utilisation is fairly high (and therefore provides a good basis for fault injection). As before, R_i in the table refers to the (hard) worst case response time, calculated from equations (2.5) and (2.6) without any faults. Additionally, the table shows H_i, the hyperperiod at level i and a_i, the number of invocations in the hyperperiod at level-i (§3.4.1).

In the same way as for the harmonic set, it is useful to consider the pattern of worst case response times for one fault within the hyperperiod, see Figure 3.6. There are 7875 invocations of frame 8 in the hyperperiod; for clarity, the graph shows only

[5] Again, assuming worst case bit-stuffing.

Table 3.2: Non-harmonic Message Set.

Pri	Bytes	C_i (ms)	T_i (ms)	J_i (ms)	D_i (ms)	R_i (ms)	H_i (ms)	a_i
17	62	0.496	1000	0.2	5	1.616	1000	1
16	72	0.576	4.5	0.2	4.5	2.216	9000	2000
15	62	0.496	5	0.2	5	2.736	9000	1800
14	72	0.576	6	0.2	5	3.336	9000	1500
13	62	0.496	8	0.2	5	3.856	9000	1125
12	72	0.576	9	0.2	5	4.456	9000	1000
11	112	0.896	10	0.2	10	5.216	9000	900
10	62	0.496	12	0.2	10	7.456	9000	750
9	72	0.576	14	0.2	10	8.056	63000	4500
8	72	0.576	16	0.2	10	9.176	126000	7875
7	62	0.496	18	0.2	20	12.336	126000	7000
6	92	0.736	120	0.2	100	14.136	126000	1050
5	62	0.496	140	0.2	100	16.376	126000	900
4	62	0.496	160	0.2	100	18.016	252000	1575
3	82	0.656	1000	0.2	1000	18.536	252000	252
2	62	0.496	1200	0.2	1000	22.816	252000	210
1	62	0.496	1400	0.2	1000	22.84	252000	180

the first 200, but the rest of the invocations exhibit a similar trend throughout the hyperperiod. As can be seen, there is huge variation in the worst case response times; it is this variation that may be exploited to provide improved fault tolerance.

The difference between the worst-case and other scenarios can be seen clearly in Figure 3.7 where the response times are plotted cumulatively; the line on the graph represents how many messages within the hyperperiod (at level-i) are guaranteed to arrive within a given time. For example, 50% of the messages within the hyperperiod are guaranteed to arrive within 3.656 ms, only 45 of the 7875 invocations can possibly arrive after 8 ms of release and only 21 frames in 7875 (0.27%) can possibly arrive after the deadline of 10 ms.

This simple example shows that weakly-hard analysis is able to consider the behaviour at high levels, with precise indication of the pattern of deadlines that can be missed. Allowing a small number of deadlines to be missed in a flexible bus results in a system where a large, predictable number of deadlines are met.

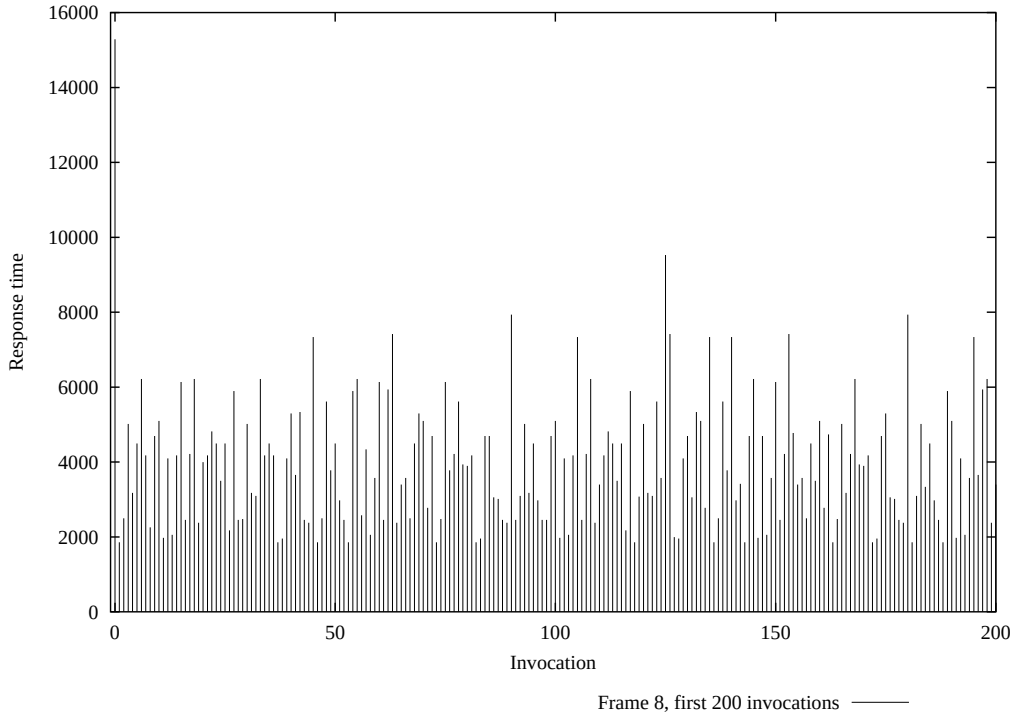


Figure 3.6: Variation in Worst Case Response Times for One Fault in Hyperperiod (only part of hyperperiod shown).

3.9.4 Hard Deadlines with High Levels of Faults

At higher levels of faults, analysis considering multiple invocations and weakly-hard deadlines has advantages over WCRT analysis and hard deadlines. To show this, fault models of $f \in \{0, 60, 80, 160, 200, 320\}$ single bit errors per second are used, assuming a minimum inter-arrival time $T_F = \frac{1}{f}$. Worst case response time analysis and then weakly-hard analysis are performed.

Normal (single invocation) worst case response time analysis was performed on the message set, using equations (2.5) and (2.10). The worst case response times are shown in Table 3.3. Note that at higher fault levels, many of the frames are not schedulable (they would arrive after their deadlines or may not arrive at all—unbounded). For even one fault in the hyperperiod, some frames (12, 9, 8) can miss their deadline in the worst case.

Using the formulation given in Section 2.5 it can also be shown that when the system

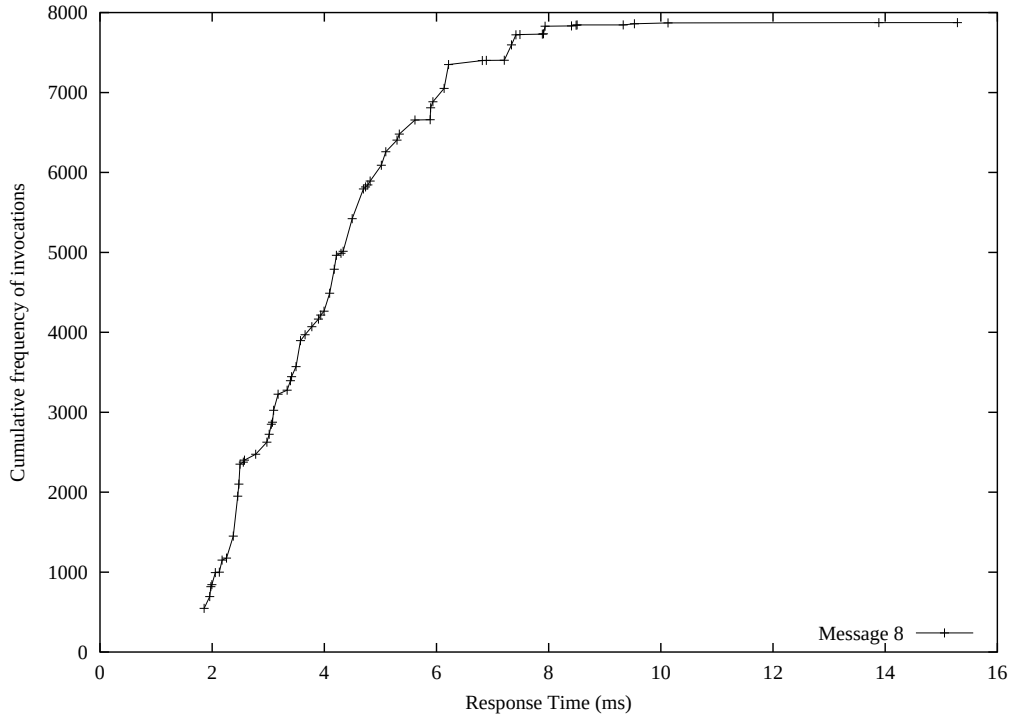


Figure 3.7: Cumulative Distribution of Response Times with One Fault (frame 8).

reaches error rates larger than $\sim$ 62 faults/second ($T_F = 16.128\,\text{ms}$) then the limit of the analysis is reached because the WCRT exceeds the period (frame 7).

3.9.5 Weakly-hard Analysis at High Levels of Faults

Although the system is not strongly-hard schedulable at greater than 62 faults/sec, it is weakly-hard schedulable. For the same fault models weakly-hard analysis was performed. As weakly-hard analysis can produce a lot of information, some of little interest, the analysis results are not presented in full, instead illustrative message streams are chosen.

Frame 12 at 60 Faults/Sec

First consider frame 12 (which misses its deadline in the worst case with even 1 fault). Hence WCRT cannot provide any guarantees. Weakly hard analysis shows that in

Table 3.3: Worst Case Response Times (ms) with Faults.

	f (faults per second)					
Pri	0	60	80	160	200	320
17	1616	2368	2368	2368	2368	2368
16	2216	3048	3048	3048	3048	3048
15	2736	3568	3568	3568	3568	4400
14	3336	4168	4168	4168	4168	5000
13	3856	4688	4688	4688	4688	+
12	4456	*6408	*6408	*6408	*7840	+
11	5216	8088	8088	9760	9760	+
10	7456	9128	9128	+	+	+
9	8056	*12368	*12368	+	+	+
8	9176	*15288	+	+	+	+
7	12336	16328	+	+	+	+
6	14136	23040	23040	36488	69464	+
5	16376	24160	24160	47632	69984	+
4	18016	26840	29792	48152	79928	+
3	18536	27360	30312	54104	89872	+
2	22816	29680	34592	60336	107600	+
1	22840	29704	34616	60360	107624	+

Key

***** Missed Deadline
+ $R > T$

the 1000 invocations per hyperperiod, with $T_F = \frac{1}{60}$ s it is only possible to miss three deadlines. The following weakly-hard guarantees can be derived:

- $\binom{997}{1000}$ in any 1000 invocations, 997 deadlines are met;
- $\overline{\langle 2 \rangle}$ two deadlines in a row are never missed.

As well as weakly-hard guarantees, it is also informative to see how frequently the weakly-hard constraints that cannot be guaranteed all of the time, may be met. For example, with reference to Table 3.4 which considers $\left\langle {n \atop m} \right\rangle$ type constraints. In any window of 6 deadlines, the analysis guarantees that 3 in a row are met; however to meet 4 deadlines in a row the analysis guarantees that this will occur at least 99.4% of the time.

Table 3.4: Percentage of Times that the Constraint $\left\langle \begin{smallmatrix} n \\ m \end{smallmatrix} \right\rangle$ is Satisfied (Frame 12, $f =$ 60 faults per second).

	n					
m	1	2	3	4	5	6
1	99.7					
2	100.0	99.4				
3	100.0	99.7	99.1			
4	100.0	100.0	99.4	98.8		
5	100.0	100.0	99.7	99.1	98.5	
6	100.0	100.0	100.0	99.4	98.8	98.2

Frame 12 at 160 Faults/Sec

At an even higher level of faults, $T_F = \frac{1}{160}$ s, frame 12 shows similar results. The worst case response times of the invocations ($R_{i,k}$) are different, but the pattern of missed deadlines is identical at 60 and at 160 faults per second. Compared to normal WCRT, weakly-hard analysis allows considerable insight into the behaviour of the system at high levels of faults, showing that even in such a disturbed environment, message delivery is analysable and (perhaps surprisingly) is far more reliable than worst case response time analysis might indicate.

Frame 7 at 160 Faults/Sec

At $T_F = \frac{1}{160}$ s, the lower priority frame, 7, cannot even be given a response time using standard WCRT analysis because the worst case response time is greater than the period.

However, multiple invocation analysis and evaluation of weakly-hard constraints shows that even this message is guaranteed schedulable in 99.24% of cases. The guarantees that are derived include:

$\left(\begin{smallmatrix} 110 \\ 112 \end{smallmatrix}\right)$ in any window of 112 there will always be at least 110 deadlines met;

$\left\langle \begin{smallmatrix} 12 \\ 25 \end{smallmatrix} \right\rangle$ in all windows of 25, at least 12 are guaranteed to be met in a row;

$\overline{\langle 3 \rangle}$ three deadlines in a row are never missed.

Additionally, considering the number of times that constraints are not guaranteed, 99.99% of the time, two deadlines in a row are always met. Table 3.5 shows the number of times that the $\overline{\langle n \rangle}$ type constraints are guaranteed to be met.

Table 3.5: Analysis of deadlines missed in a row, Frame 7 at 160 faults/second.

Constraint:	$\overline{\langle 1 \rangle}$	$\overline{\langle 2 \rangle}$	$\overline{\langle 3 \rangle}$	$\overline{\langle 4 \rangle}$
Percentage Achieved:	99.24	99.99	100	100

Frame 7 at 200 faults/second

Table 3.6 shows the percentage of times the constraint $\binom{n}{m}$ is satisfied in the worst case for frame 7 under a fault rate of 200 errors per second. Even under these extreme error conditions, the analysis shows that only 19.06% of the deadlines can be missed (since 80.94% of $\binom{1}{1}$ deadlines are met).

Analysis of the number of missed deadlines in a row, Table 3.7, shows that it is possible to miss up to 7 deadlines in a row (0.014% of the time) but that it is impossible to miss 8 in a row ($\overline{\langle 8 \rangle}$ is achieved 100% of the time).

Table 3.6: Percentage of Guaranteed Weakly Hard Constraints $\binom{n}{m}$ by Analysis, Frame 7 at 200 faults/sec.

m	n=1	2	3	4	5	6	7	8	9	10
1	80.94									
2	99.12	62.75								
3	99.92	95.18	47.71							
4	99.94	99.88	90.10	33.84						
5	99.95	99.92	98.30	80.84	25.68					
6	99.97	99.94	99.87	95.88	70.31	19.67				
7	99.98	99.95	99.92	99.78	92.30	60.24	14.40			
8	100.00	99.97	99.94	99.88	99.65	87.64	49.48	10.95		
9	100.00	100.00	99.95	99.90	99.85	98.37	80.90	41.30	8.2000	
10	100.00	100.00	100.00	99.91	99.88	99.77	96.30	74.27	33.84	5.4429

Table 3.7: Analysis of Deadlines Missed in a Row, Frame 7 at 200 faults/second.

Constraint:	$\overline{\langle 1 \rangle}$	$\overline{\langle 2 \rangle}$	$\overline{\langle 3 \rangle}$	$\overline{\langle 4 \rangle}$	$\overline{\langle 5 \rangle}$	$\overline{\langle 6 \rangle}$	$\overline{\langle 7 \rangle}$	$\overline{\langle 8 \rangle}$
% Achieved:	80.94	99.12	99.92	99.94	99.96	99.97	99.99	100

At 320 faults per second, utilisation exceeds the bus capacity ($U > 1$) and for frames 11 and below, times cannot be bounded.

3.9.6 Summary of Analysis at High Levels of Faults

Normal worst case analysis applied to the non-harmonic benchmark shows that even for low levels of faults (only one single fault) it is possible to miss a deadline. Therefore considering all deadlines to be hard, the system is not robust.

However, by considering deadlines to be weakly-hard and applying suitable analysis, this section has shown that a number of weakly-hard guarantees can indeed be made about the number of deadlines missed and the patterns of deadlines missed in a row. In particular at up to 160 faults per second, a fairly high reliability is guaranteed for all frames (the examples showed frames 7, 12 which are the most likely to miss deadlines). At 200 faults per second, frame 7 began to show serious degradation of performance, where it is possible to miss up to 7 deadlines in a row (although with a small probability). This type of information may be used by system designers to consider the reliability of their system. Two particular advantages of using this analysis over normal worst case response time analysis are:

- system behaviour is *predictable* at high levels of faults: rather than a simple yes/no indication of predictability, the minimum guaranteed performance can be understood in terms of the number and pattern of deadline misses and weakly-hard constraints;

- the levels of faults that can be usefully considered is far higher than conventional worst case response time can accommodate: in the example, above 62 faults/second, normal worst case response time analysis fails, yet the weakly-hard analysis provided useful results at 200 faults/second.

Depending on the safety requirements of the application, missing even a single deadline may not be acceptable. However, for many systems, the ability to provide guarantees of some deadlines at high levels of faults, and the investigation of ‘what if?’ scenarios, leads to a better understanding of the behaviour of the system.

3.10 Simulation of CAN

Analysis (including multiple invocation/weakly-hard analysis) is a useful tool for the verification of the timing behaviour of systems. However, as with any form of analysis, if it does not accurately reflect the behaviour of a real system, then it is of less use. In order to determine the differences between weakly-hard analysis of CAN and a real CAN system, a software-based CAN bus simulator has been constructed which simulates messages and errors on a CAN bus. It accurately simulates the effect of faults on the bus, including truncating messages, error frames and retransmissions for a variety of fault models.

3.10.1 On Fault Models and Motivation for Simulation

The sporadic fault model (based on a minimum inter-arrival time between faults) used in the previous weakly-hard analysis is the same fault model that has become standard industrial practice. Using this model is very useful for types of *worst case* analysis such as Tindell's original analysis [159] and weakly-hard analysis because it allows a maximum bound to be placed on the effect of faults. However, there are a number of problems with this model as previously discussed in Section 2.7.4.

A further problem with the bounded fault model is that it raises a difficulty in simulation: how can you simulate a sporadic fault model with a minimum-interarrival time and hope to achieve the worst case? A simulation which simply imposes faults separated by the minimum inter-arrival time would not be appropriate because this would generate a periodic fault—which is both "unrealistic" and unlikely to generate the worst case unless the periodic stream is given a carefully adjusted offset!

The answer is to consider carefully *why* the simulation is to be performed. Three aims are identified:

1. To provide confidence in the correctness of the analysis.

2. To gauge the pessimism in the analysis.

3. To test the resilience of the analysis (based on the bounded fault model) in a

more realistic scenario where faults may not adhere to the fault model used in the analysis.

The first two aims are considered together in the Section 3.10.3. The third aim of the simulation is explored in Section 3.11.

3.10.2 Simulation Model and Parameters

The following simulations are performed using a purpose-built software CAN simulator. It models CAN at the bit-level, allowing a variety of faults to be injected in a flexible way.

For the simulations: release jitter for each release is randomly added using a uniform distribution over the range $(0, J_i)$, there is perfect clock synchronisation, and maximum bit stuffing is always forced to occur. Any other characteristics of the schedule (such as the presence of blocking) are not coerced in any way.

3.10.3 Verification and Pessimism

Simulation may be used to gauge the pessimism in the analysis and to build confidence in the correctness of the analysis. Naturally, a simulation based approach cannot ever *prove* that an analysis is correct, the best that it can provide is to build confidence in the use of the analysis.

In order to verify that the analysis is correct, it is useful to recall that the weakly-hard analysis is a form of worst case response time analysis. Therefore the aim of simulation is to verify that the response times greater than the worst case are not observed. With this aim in mind, a fault model suitable for this form of simulation is chosen.

As the preceding section discussed, a periodic fault model is not appropriate, and to simulate all possible offsets of a periodic fault model is infeasible. Yet a sporadic fault model where faults occur at less than the minimum interarrival time is not the worst case. On the other hand, a random arrival model has a probability of exceeding the worst case.

For the simulations, a random fault-arrival model based on a Poisson distribution is chosen, but with the restriction that if two faults are closer than T_F then the second fault is delayed until T_F and all subsequent faults are delayed by the same amount. This way, the minimum inter-arrival time assumption is never broken, yet a periodic fault is avoided. This is not worst-case, but through extended simulation, bad scenarios should be generated.

The simulation fault model therefore has two parameters, T_F (minimum interarrival time) and λ (the parameter of the Poisson distribution: expected number of faults in a unit interval). The relationship between T_F and λ does not particularly matter provided that a high proportion of faults do occur with separation near to T_F. In the following, the parameters are chosen such that $T_F = 1/\lambda$, so that on average, half of the faults will be generated with the minimum interarrival time T_F.

Results

Simulations were performed at $\lambda = \{60, 160, 200\}$ faults per second, the resulting observed[6] weakly-hard constraints are compared with the weakly-hard analysis results in Table 3.8. Recall the notation $\binom{n}{25}$ refers to n deadlines that are met is any consecutive window of 25.

The weakly-hard constraints guaranteed by the analysis were not broken during simulation, providing confidence in the correctness of the analysis.

Regarding the pessimism in the analysis, from Table 3.8, although the analysis may predict that deadlines can be missed, simulation experiments show that even with fault rates close to the limit for weakly-hard schedulability, very few deadlines are actually missed.

Even at much higher error rates than the analysis (compare the constraints for $T_F = \frac{1}{60}$ s and $T_F = \frac{1}{160}$ s in Table 3.8) the simulation still shows that weakest observed weakly-hard constraints are stronger than the strongest constraint that is guaranteed indicating that the CAN weakly-hard analysis is pessimistic.

However, the difference between the simulation results and the analysis results is

[6] See §3.3.1.

Table 3.8: Comparison of Weakly-hard Analysis and Simulation at High Levels of Faults.

	60 f/s			**160 f/s**			**200 f/s**		
i	Ana.	Simulation		Ana.	Simulation		Ana.	Simulation	
	$\binom{n}{25}$	M/H	$\binom{n}{25}$	$\binom{n}{25}$	M/H	$\binom{n}{25}$	$\binom{n}{25}$	M/H	$\binom{n}{25}$
17	25	0.0	25	25	0.00	25	25	0.00	25
16	25	0.0	25	25	0.00	25	25	0.00	25
14	25	0.0	25	25	0.00	25	25	0.00	25
13	25	0.0	25	25	0.00	25	25	0.00	25
12	24	0.1	24	24	0.05	24	24	0.10	24
11	25	0.0	25	25	0.00	25	25	0.00	25
10	25	0.0	25	22	0.00	25	22	0.00	25
9	24	0.0	25	21	0.00	25	17	0.35	24
8	23	0.4	24	15	3.00	24	6	9.75	23
7	25	0.0	25	23	0.00	25	23	0.15	24
6	25	0.0	25	25	0.00	25	25	0.00	25
5	25	0.0	25	25	0.00	25	25	0.00	25
4	25	0.0	25	25	0.00	25	25	0.00	25
3	25	0.0	25	25	0.00	25	25	0.00	25
2	25	0.0	25	25	0.00	25	25	0.00	25
1	25	0.0	25	25	0.00	25	25	0.00	25

Key

Ana. $\binom{n}{25}$ Strongest constraint guaranteed by analysis.

M/H Average number of deadlines missed per hyperperiod in simulation.

Simulation $\binom{n}{25}$ Weakest constraint observed during simulation.

small enough to be useful. In comparison to WCRT analysis, weakly-hard analysis allows significantly more deadlines to be guaranteed and predictions of behaviour at far higher levels of faults. The causes of pessimism are discussed further in Section 3.12.

3.11 Resilience of the Analysis

Earlier, the bounded fault model was criticised for being unrealistic (it does not match real observations of faults), yet it is beneficial to use a bounded fault model because without it a worst case response time analysis is impossible. However, a worst case response time analysis based on incorrect or poor assumptions is not a strong argument for use of CAN in a dependable system.

This section will evaluate the *resilience* of the weakly-hard analysis (given that it uses a bounded fault-model) when a more appropriate fault model is applied. Since the fault model at run-time will be different to the fault model for analysis, any analysis results cannot be interpreted as *absolute guarantees of behaviour*. The aim of these experiments is to consider the appropriateness of the bounded fault model under more realistic conditions.

Simulations are performed using a more realistic fault model, the results being analysed for weakly-hard constraints which will then be compared against the guarantees made by the weakly-hard response time analysis using the bounded fault model.

The new fault model will be the simple Poisson random fault model commonly used to describe faults induced by electromagnetic interference [36, 114], where single bit faults occur according to a Poisson distribution of arrival times.

In the tests $\lambda = \frac{1}{T_F}$, that is, the average number of faults in one second of simulation will be the same as the maximum number of faults per second used for worst case analysis. In this way, scenarios are generated that are worse than the worst case that was analysed, yet overall the average number of faults injected is no worse than that used by the analysis.

3.11.1 Simulation

The same message set used in Section 3.7 (non-harmonic version of the SAE benchmark) was simulated for an interval of 5040 seconds (20 hyperperiods). Faults were injected onto the bus according to a random Poisson distribution.

At the fault rate of $\lambda = \frac{1}{T_F} = 200$ faults per second, the simulation results were analysed for *observed* weakly-hard constraints in a window of 25 deadlines. The analysis and observed constraints appear in Table 3.9. Where the simulations break a constraint from the weakly-hard analysis, an entry is shown in the table as a percentage of the number of times that the constraint was not satisfied per the number of times the constraint was evaluated.[7]

The simulations showed:

- in most cases, the analysis weakly-hard guarantees were pessimistic, note stream 8 where the analysis guaranteed only $\binom{6}{25}$, yet in the worst scenario observed, 23 in 25 deadlines $\binom{23}{25}$ were met;
- nearly all guarantees made by the weakly-hard analysis with the bounded fault model held when the Poisson fault model was applied. The constraints were broken only occasionally, note stream 11, where the analysis guaranteed $\binom{25}{25}$ (*i.e.* all deadlines are met—a hard deadline guarantee), yet one deadline was not met in this stream during the simulation; this corresponds to a failure of the guarantee with a probability of $5 \cdot 10^{-5}$ (although the length of the simulations is not sufficient to derive this figure with any accuracy).

As shown in the table, the CAN simulation usually performs better than the weakly-hard analysis predicted. The number of times that the constraints are broken is extremely small (one constraint broken in the whole simulation). This indicates that the simple bounded fault model is not wholly inappropriate and that its use in an analysis such as weakly-hard analysis is useful and justified. If the bounded fault model is

[7] Note that where there is one single deadline missed in the whole simulation run, this corresponds to 25 $\binom{25}{25}$ constraint failures. This is because the constraint will be tested in all possible positions—the size of the constraint window is 25 frames and therefore it can be placed in 25 different ways over the one missed deadline, one missed deadline is counted a total of 25 times.

Table 3.9: Comparison of Weakly-hard Constraints ($T_F = \frac{1}{200}$ s) and Poisson Simulations at $\lambda = 200$ faults/s.

	Analysis		**Simulation**	
	WCRT	Strongest Guarantee	Weakest Observed	Guarantee Broken
i	R_i (ms)	$\binom{n}{25}$	$\binom{n}{25}$	%
17	2.368	25	25	
16	3.048	25	25	
15	3.568	25	25	
14	4.168	25	25	
13	4.688	25	25	
12	*7.840	25	24	
11	9.760	25	24	0.005
10	+	22	25	
9	+	17	24	
8	+	6	23	
7	+	14	24	
6	69.452	25	25	
5	69.972	25	25	
4	79.900	25	25	
3	89.928	25	25	
2	107.624	25	25	
1	107.648	25	25	

Key

+ $R_i > T_i$

***** $R_i > D_i$ (Missed Deadline in Hard Worst Case Analysis)

Guarantee Broken % shows the percentage of times that a constraint was false. It is shown only when the simulation gave worse results than the analysis.

considered a useful approximation (rather than a perfect model) then the weakly-hard analysis results are an extremely useful (and accurate) description of the behaviour of the system at high levels of faults.

3.12 Pessimism in Weakly-hard Analysis

As the previous sections showed, the weakly-hard constraints produced by analysis were generally stronger than the constraints observed during the simulations. This was also true when a more realistic fault model is applied, although occasional circumstances are expected where the analysed constraints were weaker than observed during simulation.

The reasons for this effect are both pessimism in the analysis *and* the worst case conditions in the analysis rarely occurring during simulations. There is an important difference between pessimism in the analysis (describing scenarios which *cannot* occur) and unlikely scenarios which *could* occur but were not observed during simulation.

True Pessimism

The main source of pessimism concerns the worst case scenarios for faults. As explained in Section 3.5 each invocation has a different worst case scenario, described as an offset. This worst case scenario is used to determine the worst case response time that each invocation may have. However, it may not be possible for the worst case scenarios for two different invocations to occur within the same hyperperiod because the offset implied by one worst case scenario is different to the offset implied by the other. Therefore the two invocations *may not* both be able to miss their deadlines in the same hyperperiod even though the analysis has determined that both invocations *may* miss their deadlines.

Unlikely Scenarios

There are many unlikely scenarios: situations that are unlikely to occur, but nevertheless could occur. These scenarios are the main contribution to the difference between analysis and simulation results.

As the analysis is a worst case analysis, it uses the fault overhead of the worst possible fault for all faults. In this case, it assumes that the fault occurs on the last bit of a frame, so that the maximum amount of bandwidth is used before retransmitting the frame. In the simulations, this is not the case as faults may occur anywhere within the frame; the simulator accurately terminates the frame at that point.

It could also be said that this is a true form of pessimism because (except in contrived circumstances) there will be faults which cannot occur on the last bit of a frame because the sporadic fault arrival time T_F will fail to coincide with the end of frames. However, the fact that there is a discussion on this merely indicates that the bounded fault model has limitations in analysis.

Another unlikely scenario is the assumption that a frame receives maximum blocking at each invocation. In this sense, the analysis is equivalent to a message set in which there is one additional message in the bus at higher priority, whereas in a real bus, and the simulation, maximum blocking rarely occurs.

Additional causes of unlikely scenarios (such as bit stuffing distributions [118]) also contribute to the difference between analysis and a real-world bus (although the simulations assumed maximum bit stuffing). The topic of considering unlikely scenarios is returned to in the next chapter.

3.13 Summary

This chapter has explored the concept of missing occasional deadlines in real-time communication, in an analysable way, through the use of weakly-hard constraints and analysis. In practical systems, where some deadlines may be missed [116, 21] many "hard" real-time systems are actually able to be expressed as weakly-hard systems. On this basis, the specific contribution of this chapter is the application of weakly-hard

schedulability analysis to CAN.

Weakly-hard analysis exploits the flexibility of event-triggered communication by considering multiple invocations of a message frame, rather than just considering the worst case. Weakly-hard analysis can show that even though a deadline may be missed in the worst case, in many other cases (other invocations), it is not possible to exceed the deadline.

Therefore, weakly-hard analysis is able to predict behaviour at much higher utilisation, including higher levels of faults than usual worst case response time analysis can consider. The analysis provides information about exactly which deadlines may be missed, rather than merely the pass/fail schedulability test that is normally used.

In this way, the analysis provides a useful tool for considering 'what if?' scenarios, allowing behaviour to be predicted for possible exceptional circumstances, as well as allowing better use of bandwidth under normal circumstances.

The advantages of using non-harmonic periods were explained. The increased fault-tolerance that non-harmonic periods provide was shown through a case study. By introducing non-harmony, the hyperperiod is extended, thus the frequency of occurrences of scenarios which may lead to a deadline miss is reduced.

Case studies were used to compare normal worst case response time analysis with weakly-hard analysis, and also with simulation. The weakly-hard analysis was shown be usefully pessimistic with respect to the simulations, yet provide tremendous advantages over worst case analysis (in the sense of predicting behaviour at high levels of faults).

The bounded fault model used for any worst case response time analysis (including weakly-hard analysis) has briefly been discussed and its benefits expressed. Although there is doubt about its ability to represent real-life faults, simulations have shown that when a more realistic fault model is applied to CAN, weakly-hard guarantees can be broken, but this is observed to occur only rarely (see Table 3.9). This is addressed further in Chapter 4. Generally, the weakly-hard analysis is sufficiently pessimistic to show resilience to a more realistic Poisson model, and the use of the bounded model is justified in this form of response time analysis.

4 Probabilistic Analysis of CAN

MISSING DEADLINES IN AN ANALYSABLE WAY is an acceptable and useful method of considering the effects of network faults. The previous chapter showed that if a small number of deadlines can be missed then far higher levels of faults can be tolerated in a predictable manner. This implies that rather than requiring a high temporal redundancy to guarantee all deadlines, large savings may be made if some deadlines are missed.

This chapter continues this theme further, showing not only that missing deadlines can be beneficial, but that it may not be *possible* to avoid missing deadlines in communication in practical situations. Therefore, a move from absolute guarantees to a probabilistic notion is natural.

In this chapter, some problems of worst case analysis and the *bounded fault model* are explained and (despite its use in weakly-hard analysis) it is found to have limitations. A probabilistic fault model is introduced instead which is considered to be a more realistic scheme. This requires a new analysis approach that instead of providing guarantees, produces probabilistic predictions of response time behaviour. The main contribution of this chapter is a new form of analysis, which is effective at dealing with unlikely scenarios and yet avoids the pessimism associated with both traditional worst case response time analysis and with similar probabilistic work.

4.1 Worst Case Analysis and Fault Models

This chapter begins with a short discussion on the problems of worst case analysis to provide motivation for alternative fault models and a probabilistic response time

analysis. There are two main failings of the usual worst case response time analysis: the use of a *bounded failures assumption* and often a resulting *poor utilisation*.

4.1.1 Bounded Failures

From Lemma 1 on page 59, in order to be able to do any form of worst case analysis, it is necessary to be able to determine the worst case scenario(s). Specifically, for CAN, to consider the worst case response time of a particular frame it is necessary to determine the worst case interference, blocking and overhead due to faults.

Applied to the familiar worst case response time equation from page 66, repeated here:

$$t_i = B_i + C_i + I_i(t_i) + E_i(t_i) \tag{2.10}$$

it is necessary to determine the absolute maximum value of $E_i(t)$.

This means that no matter what form of fault model is used, it must be a *bounded* fault model—that is, there must be some bound on the maximum overhead due to faults. The common way to impose such a bounded fault model, as the previous chapter did, is to assume a sporadic fault arrival model [159].

However, a mathematical convenience is no justification for basing an analysis on such a model. It is necessary to consider whether or not a minimum inter-arrival time does exist in reality if we are to have any confidence in the analysis.

In general, there is little evidence that this is how faults occur (in any domain, not just electrical interference). It would be a fallacious argument to state that just because a computer crashed yesterday, it cannot crash today; likewise with electrical interference on CAN, the previous message being corrupt, *does not ensure* that the next one is not corrupt. It is clear that nature does not *guarantee* that faults cannot occur closer together than some bound, even if it may rarely be observed.

The same argument is true of any other bounded fault model,[1] for example assuming a bound on the maximum number of faults within an interval.

[1] Except the bounded fault model that states that data on the bus is always corrupt: there is no worse scenario so the bound is truly a bound. However, this bus has little practical use of course.

Since there can be no *guarantee* of minimum fault inter-arrival times (or any other bounded measure), there is no bound on the overhead due to faults. Therefore there can never be an *absolute* guarantee of worst case response times. In other words, although for most of the time, worst case response times may be accurate, they cannot be absolute guarantees (as demonstrated by the simulations in Chapter 3). This is an important concept because to rely on 'guarantees' made by worst case analysis of CAN is not automatically safe.

Therefore, despite the variety of possible uses of a bounded fault model (weakly-hard analysis for example) a worst case response time analysis is not possible if one is to have *absolute* confidence in the results. This is a fundamental problem of using assumptions about bounded faults.

4.1.2 Poor Utilisation

Suppose that a bounded fault model is used, despite the problems described in the previous section; a minimum inter-arrival time is set to a suitably small value that there is some confidence in the assumption. Then (as the previous chapter also discussed), worst case response time analysis can lead to poor utilisation because the value of $E_i(t)$ becomes so large that the message set rapidly becomes unschedulable according to worst case response time analysis—even though observations in practice show that deadlines are rarely missed, if at all.

This is a fundamental problem with a single valued response time analysis, (which was partially solved in the previous chapter by considering multiple invocations). A complementary approach is considered in this chapter.

4.1.3 Fault Diversity and Modelling

The sources of EMI are diverse and difficult to model. It is clear that electrical interference is difficult to characterise; each application will experience different levels and types of interference and it is very hard to predict future interference with accuracy. Nevertheless, if there is to be any form of analysable, working and reliable system then there must be some sort of fault model and it must be accepted that it is not per-

fect. Section 2.7 discussed a variety of different models for electrical interference on communication channels, each model with its own merits. The sporadic model [159], the model for specific sources [131], and the bounded omission degree assumption [167] are examples of bounded fault models.

However, one fault model frequently used to describe EMI in other domains is to model faults as a *random pulse train* following a Poisson distribution [36, 80]. This model is well suited to modelling faults in electronic circuits and networks.

Navet [114] adopted a probabilistic fault model for CAN (see Section 2.7.5) using a Poisson distribution. The fault model is modelled as a stochastic process which considers both the frequency of the faults and their gravity. In that model, faults in the channel occur following a Poisson law and can be either single-bit faults (which have a duration of one bit) or burst errors (which have a duration of more than one bit) according to a random distribution.

Note that if the occurrence of faults in the channel follows a Poisson distribution, the maximum number of transmission errors suffered by the system in a given interval is not bounded, so the probability of having sufficient interference to prevent a message from meeting its deadline is always non-zero; therefore every system is inherently unschedulable (as should be expected).

A probabilistic model encapsulates the idea that there may normally be a low level of faults—well separated with large inter-arrival times—yet occasionally, the system may experience higher loads with faults occurring closer together. Such fault models have been suggested (as this section and Section 4.2 explains), however an effective response time analysis technique has been lacking. In this chapter such a technique is proposed, but first it is useful to look at some previous approaches with their advantages and their deficiencies.

4.2 Probabilistic Analysis Approaches

In this section, two previous approaches to probabilistic analysis are explored. One is aimed directly at CAN, and one is for a general fixed priority system. For the one not designed for CAN, this section shows how it can be adapted for CAN. To compare

them, the techniques are applied to the SAE benchmark from Chapter 3, Table 3.1. It is a minor contribution of this thesis that where appropriate, these approaches are modified, extended or corrected in this section and contrasted in a quantitative way.

4.2.1 Probabilistic Guarantees of Failure

Burns *et al.* [38] introduced a scheme for providing probabilistic guarantees in a fault tolerant system based on fixed priority scheduling with a random fault arrival time. The 'probabilistic guarantee' here does not mean *e.g.* '99.95% of deadlines will be met', rather it means *e.g.* 'all deadlines will be met with a probability of 99.95% during some period of operation.'

The scheme is in two parts consisting of:

1. Sensitivity analysis to determine the minimum fault inter-arrival time, T_F, that will guarantee schedulability.

2. Then, making the assumption that if faults occur further apart than T_F then the system is schedulable, else it is not, the probability is calculated that a random (in this case Poisson) fault model would generate two faults closer together than T_F during the lifetime, L, of the system.

Hence, the probability that the system fails during the lifetime of the system can be found.

CAN Sensitivity Analysis

The sensitivity analysis can be performed using the CAN WCRT equations introduced in Chapter 2: equations (2.5), (2.10) and (2.11), repeated below with $n_{burst} = 0$ as in Chapter 3 equation (3.6), are used to determine the maximum value for T_F that makes the system schedulable, for example by performing a binary search between 0 and $\frac{D_i}{\max\limits_{j \in \mathsf{hep}(i)} C_j + E_{max} + S}$.

$$R_i = J_i + t_i \tag{2.5}$$

$$t_i = B_i + C_i + I_i(t_i) + E_i(t_i) \tag{2.10}$$

$$E_i(t) = \left\lceil \frac{t}{T_F} \right\rceil \max_{k \in \mathsf{hep}(i)} (C_k + E + S) \tag{3.6}$$

Probability Calculation

Probability calculation is not dependent on the underlying scheduling model, only on the fault model. Therefore, the calculation could (in theory at least) be applied directly to CAN.

The approach by Burns [38] gives an exact solution to the problem of determining $p(W < T_F)$ (probability that two faults occur closer than T_F in the lifetime L, W is the shortest interval between two faults). However, the equation is not easy to evaluate because it includes a large summation and very large factorials and powers.

Therefore, the paper also gives two other results which represent upper and lower bounds within which the exact probability must fall. Although these bounds can be useful (particularly the upper bound, as this represents a 'safe' value), a single probability of failure is clearly preferable.

As it is not feasible to calculate the exact value, with a slight change in approach an approximation can be obtained. So, in this section three different methods are suggested to provide an approximate value for the probability of failure.

Average The mid-point between the upper and lower bounds.

Monte Carlo Simulation As the inter-arrival times of a Poisson distribution form an exponential distribution, a large number (say 100,000) of random numbers may be generated in an exponential distribution which can be compared with the value of T_F.

Exponential Approximation Taking the equation for the exponential distribution, integration between 0 and T_F gives the probability that the next fault is closer than T_F as $1 - e^{-\lambda T_F}$. Therefore each fault can be considered as an event of a binomial distribution so that the probability of failure in a lifetime is $(e^{-\lambda T_F})^n$. Making the assumption (this causes the approximation) that there are $n = \lambda L$ faults in the lifetime of the system, then $p(W < T_F) \approx (e^{-\lambda T_F})^{\lambda L}$

Comparison of Approximations

The three techniques suggested for generating a single value for the probability of failure (average, Monte Carlo simulation and exponential approximation) are applied to the SAE benchmark (Table 3.1, page 91) and plotted in Figure 4.1. As the benchmark has a very high utilisation the probability of missing a deadline (as calculated by this approach) is actually quite high, so in order to have interesting results, a lifetime (L) of only one minute is considered, and an extremely low fault rate of $\lambda = 1$ fault per second used.

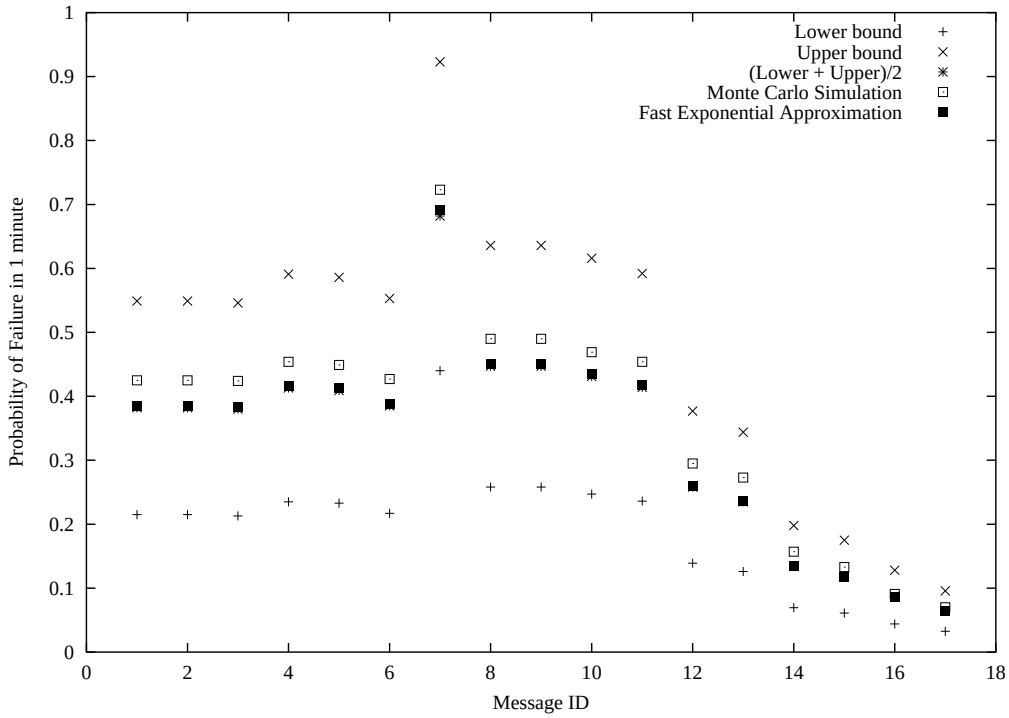


Figure 4.1: Approximations to Burns' Approach: Comparison.

As the graph shows, there is very little to choose between any of the approximations. We may consider the Monte Carlo simulation to be the most accurate of the three, but (with the exception of computing the exact value) it is the most computationally expensive. The mean and exponential approximations are very close.

When considering that the aim of the analysis is to provide an idea of the probability of failure, it is clear that an approximate value is all that is required. Therefore, it may

be concluded that any of the approximations introduced in this chapter are sufficient.

Application to Benchmark

To give some measure of the usefulness and pessimism of the approach, the results when the analysis is applied to the SAE benchmark appear in Table 4.1, with simulation results for comparison. The values for probability of failure are from the exponential approximation method.

Table 4.1: Burns and Punnekkat Guarantees vs. Simulation Results ($\lambda = 1\,\text{fault}/\text{s}$, lifetime $L = 1\,\text{minute}$).

	Analysis		**Simulation**
i	T_F	$p(Failure)$	$p(Failure)$
17	1106	0.0642	0
16	1504	0.0863	0
15	2100	0.118	0
14	2400	0.134	0
13	4488	0.236	0
12	5000	0.259	0
11	9008	0.418	0
10	9528	0.435	0
9	10000	0.451	0
8	10000	0.451	0
7	19568	0.691	0
6	8183	0.388	0
5	8879	0.413	0
4	8990	0.417	0
3	8040	0.383	0
2	8101	0.385	0
1	8101	0.385	0

As the results in Table 4.1 show, the guarantee is very pessimistic when compared with a realistic simulation of the same fault conditions. During the simulation run of 200 minutes, $\lambda = 1\,\text{fault}/\text{s}$, not a single deadline miss was observed, whereas the analysis predicted that *e.g.* message 7 would miss at least one deadline per minute in 69% of cases.

The cause of the pessimism is due to the assumption that two faults occurring close together cause a failure. Whereas in CAN, although two faults closer than T_F *may*

cause a failure, this is unlikely. The cause of failure in CAN is *more faults* rather than *close faults*. Close faults is a necessary but not sufficient condition for more faults.

Because of the extreme pessimism, this approach of considering T_F may be rejected as the basis for a probabilistic analysis on CAN.

4.2.2 Worst Case Deadline Failure Probability

Navet [114], using a scheme similar to Burns [38] produced a probabilistic response time analysis for CAN. By considering the number of faults that must occur to cause a failure (rather than how close they must occur), the pessimistic assumption in the previous section is tightened.

As before, there are two stages:

1. Sensitivity analysis to determine the minimum number of faults K_i that must occur while a frame is queued in order for the frame to miss its deadline.

2. Then, making an assumption that if there are fewer than K_i faults during this interval then the system is schedulable, else it is not, they calculate the probability that a Poisson fault model would generate more than K_i faults during the response time of a frame considering K_i faults.

Therefore, the probability of any given frame missing its deadline is known, termed *Worst Case Deadline Failure Probability* (WCDFP).

This is a slightly different result to the analysis in Section 4.2.1: Burns gives the probability of failure in a lifetime, whereas Navet gives the probability of failure per invocation. If failures are assumed to be independent (which is already an implicit assumption because of the use of the Poisson distribution) then the probability of failure per invocation can be converted to probability of failure per lifetime L as follows: consider each invocation as an event in a binomial distribution; let p be the probability of missing deadline per invocation, and $n = \left\lceil \frac{L}{T_i} \right\rceil$ be the number of invocations in a lifetime. Therefore the probability of 0 failures in a lifetime is $(1-p)^n$.

The fault model considered by Navet [114] is more complex than that of the simple single bit fault model used previously in this thesis because it considers not only the

frequency of faults, but the duration of the faults. Both the frequency and gravity are considered to follow Poisson distributions, which allows the overhead of the faults to be considered as a generalised Poisson distribution.

In more detail, Navet's analysis first uses the scheduling analysis of Tindell to calculate the maximum number of faults that can be tolerated for each message before the deadline is reached. This is done by considering that each fault generates a known maximum overhead which extends the response time. Once the maximum number of faults (K_i) and the worst case response time that this would generate,[2] $R_{i|K_i}$, are obtained, they are used in the second stage of the analysis with the fault model to find the probability that a message may miss its deadline.

Navet defines the WCDFP of a message i as the probability that more than K_i faults occur during $R_{i|K_i}$. K_i can be interpreted as the number of tolerated faults in the channel; it is independent of the parameters of the fault model, it only depends on the characteristics (length, priority, period, etc.) of the messages. Once K_i is known, the fault model parameters are required to calculate the probability of having more than K_i faults in the analysed interval. Since the fault model assumed by Navet is a generalised Poisson process, the WCDFP can be analytically calculated from K_i.

There are two main drawbacks of the analysis. Firstly, the complex fault model used means that calculating the probability of failure is difficult due to both the complexity of the mathematics and the computation time required. Secondly, the analysis includes a number of sources of pessimism, some of which dramatically increase the pessimism in the estimation of the WCDFP.

- The first source of pessimism is implicit in the definition of WCDFP which does not properly reflect the conditions in which a message can miss its deadline. For CAN, in order to miss a deadline, faults in the channel are required to occur while the message is queued or in transmission; a fault occurring after the message has been received cannot delay the message. This condition is more restrictive than the condition used in the analysis, which is that K_i errors occur at any time during the maximum response time of the message, independently of whether the message actually takes as long as $R_{i|K_i}$, *i.e.* whether the message

[2] Navet uses the notation $R_{m_{max}}$ for what this thesis terms $R_{i|K_i}$.

has already been received. This will impose even more pessimism if multiple invocations are considered because the response time for most invocations is much less than the worst case response time stemming from a critical instant (see Section 3.2).

- The second source of pessimism is an incorrect assumption about the nature of burst errors where a fault causes a sequence of bits to be corrupted. In Navet's analysis, a burst error of duration u bits is treated as a sequence of u single bit faults, each causing a maximal error overhead (an error frame and the loss of the longest data frame). This assumption allows consideration of the fault model as a generalised Poisson process and facilitates solving the WCDFP equation. However, this assumption is inconsistent with the CAN protocol specification [28, 72] since in reality a burst error can interfere with only one message because no message is sent again until the effect of the burst is finished. This pessimism may cause response times several orders of magnitude too large.

In addition, the following less severe considerations are not taken into account, which add pessimism to the analysis:

- The analysis considers only the worst case invocation, *i.e.* the first invocation following a critical instant.
- Maximal overhead of faults. The analysis assumes that each fault causes the maximal overhead, whereas this is unlikely. This is the same effect discussed previously in Chapter 3.
- Other maximal assumptions such as assuming maximal bit-stuffing and blocking have some pessimistic effect.

4.2.3 Modifications and Application to Benchmark

The pessimism caused by the fault model is excessive, leading to meaningless results. However, it is appropriate to consider the fault model separately to the overall approach, so in this section the assumption regarding multiple faults is corrected and the second stage of the analysis is adjusted accordingly. This allows the analysis to be

usefully applied directly to the SAE CAN benchmark and compared with simulation results.

Correcting WCDFP Analysis

The main difficulty with the fault model is the assumption that each bit of a *burst error* causes the same maximal fault overhead. Therefore, the complex fault model will be discarded, to be replaced with a more manageable fault model. It will be assumed that faults occur with a Poisson distribution, each fault affects a single bit and is detected immediately, causing the maximum possible overhead that a single bit fault can cause.

This new fault model does not require any changes to the first stage (sensitivity analysis) of the analysis.

The second stage of the analysis (calculating the probability that more than K_i faults occur in $R_{i|K_i}$ is modified. It is simply the application of the Poisson probability density function.

$$p(WCDF)_i = 1 - \sum_{k=0}^{K_i} \frac{e^{-\lambda R_{i|K_i}} (\lambda R_{i|K_i})^k}{k!} \tag{4.1}$$

In general, this can be easily calculated because K_i will tend to be fairly small so neither the summation nor the factorial in equation (4.1) create computational difficulties.

Application to Benchmark

The new analysis is applied to the SAE benchmark using this simplified fault model. A value of $\lambda = 30$ faults per second is used; this is much higher than in Section 4.2.1 because this analysis is far less pessimistic. Also, unlike in the previous section, the probability of missing a deadline per invocation is shown, rather than the probability of failure per lifetime; Section 4.2.2 showed how to convert between them if desired.

Simulation and analysis results are in Table 4.2 and plotted in Figure 4.2; note the logarithmic scale. As can be seen, the analysis and simulation results match closely in some cases, however for some messages (*e.g.* 7–12), the analysis predicts the prob-

Table 4.2: Navet WCDFP Analysis.

	Analysis			Simulation
i	K_i	$R_{i\|K_i}$	$p(Failure)$	$p(Failure)$
17	4	4624	$3.814 \cdot 10^{-07}$	0.000
16	3	4712	$1.486 \cdot 10^{-05}$	$2.333 \cdot 10^{-06}$
15	2	4400	$3.473 \cdot 10^{-04}$	$8.333 \cdot 10^{-06}$
14	2	5000	$5.029 \cdot 10^{-04}$	$1.033 \cdot 10^{-05}$
13	1	4688	$9.010 \cdot 10^{-03}$	$1.333 \cdot 10^{-05}$
12	0	4456	$1.251 \cdot 10^{-01}$	$1.023 \cdot 10^{-04}$
11	1	9208	$3.180 \cdot 10^{-02}$	$2.467 \cdot 10^{-05}$
10	1	9728	$3.514 \cdot 10^{-02}$	$5.000 \cdot 10^{-05}$
9	0	9176	$2.406 \cdot 10^{-01}$	$3.113 \cdot 10^{-04}$
8	0	9776	$2.542 \cdot 10^{-01}$	$2.453 \cdot 10^{-03}$
7	1	19768	$1.196 \cdot 10^{-01}$	$4.400 \cdot 10^{-04}$
6	12	99680	$1.563 \cdot 10^{-05}$	$3.333 \cdot 10^{-05}$
5	11	99048	$6.531 \cdot 10^{-05}$	$4.000 \cdot 10^{-05}$
4	11	99568	$6.857 \cdot 10^{-05}$	$4.000 \cdot 10^{-05}$
3	124	999944	0.000	0.000
2	123	999312	0.000	0.000
1	123	999336	0.000	0.000

ability of deadline failure at over an order of magnitude higher than observed during the simulations. It is unclear which particular sources of pessimism are predominant.

4.3 A New Probabilistic Analysis of CAN

In the previous section, two different approaches to probabilistic guarantees were applied to CAN. The output of both techniques was a probability of failure. However, both approaches were pessimistic when compared to simulations, several sources of pessimism were highlighted.

Furthermore, despite the advantages of these approaches, there is still a need for a more general scheme which can describe the complete behaviour of the system in probabilistic terms, rather than just dealing with deadline failures.

In this and subsequent sections, an analysis is developed which provides a probability distribution of worst case response times under a random arrival fault model.

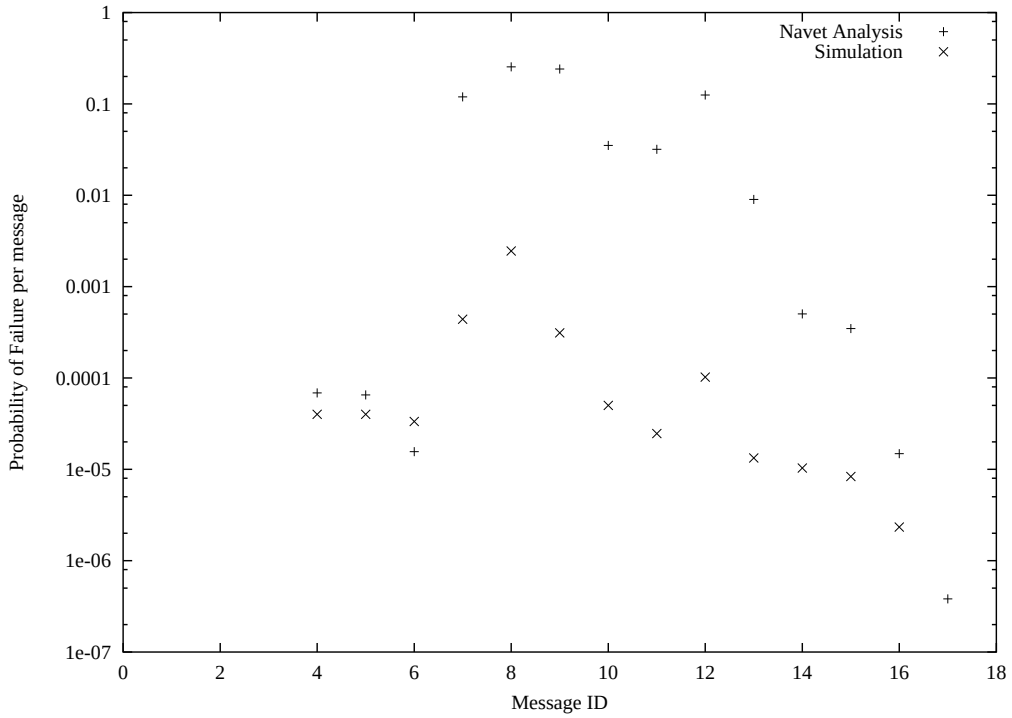


Figure 4.2: Modified Navet Analysis and Simulation Results.

This in turn may be used to determine an accurate probability of deadline failure. The derivation first produces a general analysis technique which may be applied in many situations but which has a high computational cost. In later sections, this analysis is improved for the specific CAN fault model, leading to an efficient tool.

4.3.1 Pessimistic Approach

In the context of this work (dependable real-time control), an analysis should never be optimistic—there must be no opportunity for the analysis to give a lower response time or lower probability of failure than the system it is modelling. Therefore, in each stage of the analysis, there must be no optimistic assumptions. The aim is to have a pessimistic solution, but with as little pessimism as possible. Both approaches earlier in this chapter also took this approach.

Therefore, the analysis presented in this section is derived by starting with the strict worst case analysis and removing sources of pessimism where possible. There are

several areas in the analysis where a worst case is still assumed. It would be inappropriate to assume for example a mean or other expected value for any parameter, otherwise it would risk being optimistic.

4.3.2 Fault Model

A simple fault model is considered, that faults arrive randomly, with a Poisson distribution $F \sim \mathscr{P}o(\lambda)$. Therefore, the probability of exactly m faults occurring in any time interval t is:

$$\mathrm{p}_t(F = m) = \frac{e^{-\lambda t}(\lambda t)^m}{m!} \tag{4.2}$$

To avoid optimism, it is assumed that each fault causes the maximum length error frame (E) and occurs on the last bit of the longest frame, such that the overhead due to one fault is equal to:

$$M_i = E + \max_{\forall j \in \mathsf{hep}(i)} C_j$$

where $\mathsf{hep}(i)$ is the set of messages with higher or equal priority to i.

Therefore the error overhead function, $E_i(t)$, is a random distribution:

$$E_i(t) = \begin{cases} 0 & \text{with probability } \mathrm{p}_t(F = 0) \\ M_i & \text{with probability } \mathrm{p}_t(F = 1) \\ 2M_i & \text{with probability } \mathrm{p}_t(F = 2) \\ 3M_i & \text{with probability } \mathrm{p}_t(F = 3) \\ \cdots & \cdots \end{cases} \tag{4.3}$$

or more generally:

$$E_i(t) = mM_i \text{ with probability } \mathrm{p}_t(F = m) \tag{4.4}$$

It should be noted that this error function is always pessimistic—$E_i(t)$ is always at least as great as would occur in a real bus experiencing a Poisson distribution of faults.

There is an infinite number of values for $E_i(t)$ since $m \in 0..\infty$. However, small values of m are neglected where $\mathrm{p}_t(F = m) < \rho$ and ρ is some suitably small probability

that is so small that designers consider the risk acceptable, for example $\rho = 10^{-12}$.

Therefore all 'significant' values of m may be enumerated, each case is considered individually with its associated probability.

4.3.3 Response Time—General Analysis

The analysis will consider only the critical instant where all higher priority messages are simultaneously queued (imposing maximum interference) and the longest lower priority message has just started (imposing maximum blocking). This is the known worst case scenario, so the analysis may begin with the usual worst case response time equations for CAN [159], see Section 2.5.

$$R_i = J_i + t_i \tag{2.5}$$

$$t_i = B_i + C_i + I_i(t_i) + E_i(t_i) \tag{2.10}$$

Equation (2.10) may calculated iteratively in the usual manner using a recurrence relation:

$$t_i^{n+1} = B_i + C_i + I_i(t_i^n) + E_i(t_i^n) \tag{4.5}$$

with $t_i^0 = C_i$. Iteration terminates when $t_i^{n+1} = t_i^n$ provided $t_i^n \leq T_i - J_i$.

When equation (4.5) is solved iteratively, it generates a monotonically increasing set of values of t_i and therefore a set of non-overlapping intervals exists:

$$\{(0, t_i^0],\ (t_i^0, t_i^1],\ (t_i^1, t_i^2],\ \cdots\ (t_i^k, t_i^k]\}$$

Within each interval, the error overhead function, $E_i(t)$, is evaluated for each significant value of $E_i(t_i^n - t_i^{n-1})$ and is repeated recursively for all values of t_i^n.

In order to do this, a probability tree is used. An example is shown in Figure 4.3 which is used in the following walk-through.

Using equation (4.5) and the initial value $t_i^0 = C_i$ the set is considered which consists of all possible numbers of faults in the interval $(0, t_i^0]$ for which there is a significant

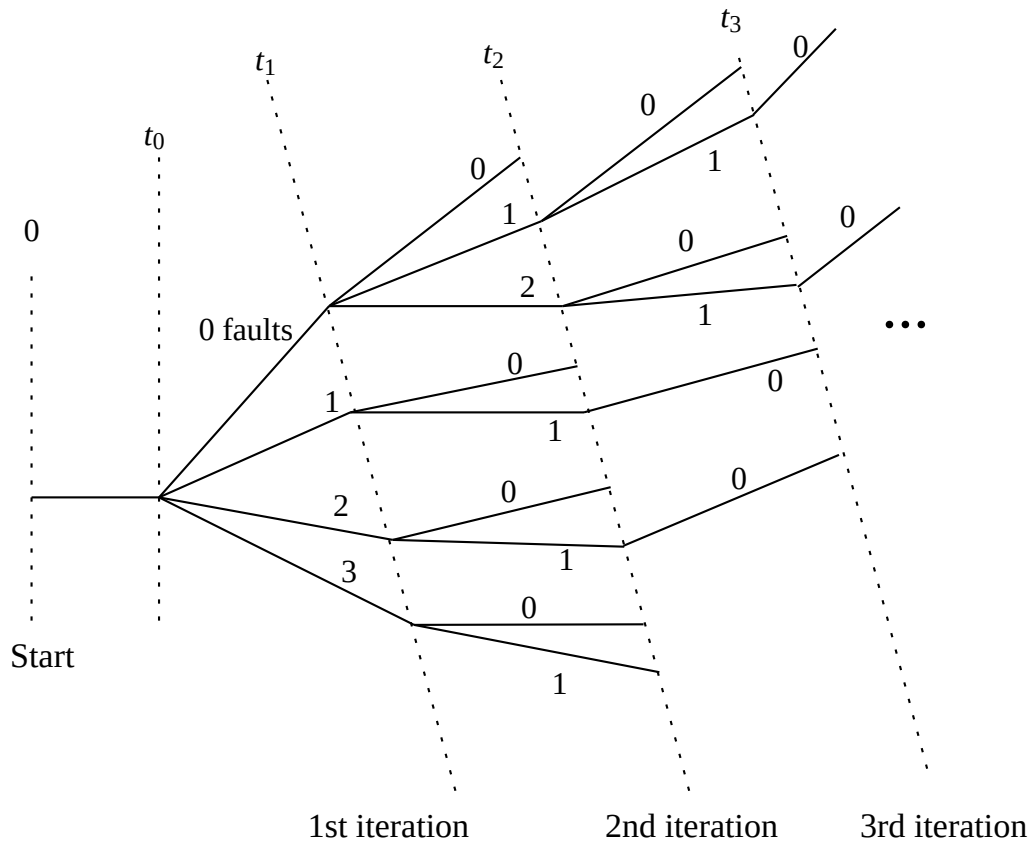


Figure 4.3: Example Probability Tree for Simple Fault Model.

probability.

$$A_{t_i^0} = \{ f | \mathrm{p}_{t_i^0}(F = f) \geq \rho \} \tag{4.6}$$

Each member of $A_{t_i^0}$ is used to derive a set of possible values for t_i^1 with an associated probability $\mathrm{p}(t_i^1) = \mathrm{p}_{(t_i^0 - 0)}(F = f)$. In the illustration there are four significant possibilities: in the interval there could be 0, 1, 2 or 3 faults, (the probability of more than 3 faults is less than ρ) so four branches of the tree are derived, with 4 different values of t_i^1 and associated probabilities.

Following the first branch (0 faults), next consider the interval $(t^0, t^1]$ and apply the Poisson equation (4.2) to the interval $(t^0, t^1]$. (This may be done because the probability of any number of faults in an interval is independent from the history of faults before the start of the interval.) Therefore we apply equation (4.2) to determine that there are three *significant* possibilities: there are 0, 1 or 2 faults in the interval $(t_0, t_1]$.

So the three branches provide three possibilities for t_i^2 and associated probabilities of $\mathrm{p}(t_i^2) = \mathrm{p}(t_i^1) \cdot \mathrm{p}_{(t_i^1 - t_i^0)}(F = f)$.

The tree continues to be explored recursively. For subsequent iterations of equation (4.5), the error overhead function must be applied with care. At each branch in the tree, the path to the root of the tree has been fixed, therefore when $E_i(t)$ is evaluated, it must only consider the time between t_i^n and t_i^{n-1}. Yet it must not neglect the previous error overhead. The formulation may be written:

$$t_i^{n+1} = B_i + C_i + I_i(t_i^n) + \sum_{j=0}^{n-1} E_i(t_i^{j+1} - t_i^j) \tag{4.7}$$

However, equation (4.7) is easier to understand and implement if written as:

$$t_i^{n+1} = B_i + C_i + I_i(t_i^n) + E_i^n \tag{4.8}$$

and E_i^n is the total error overhead for the previous iterations in the path to the root of the tree, plus the overhead added this iteration:

$$E_i^n = E_i^{n-1} + E_i(t_i^n - t_i^{n-1}) \tag{4.9}$$

Evaluation continues until all significant branches are explored. The base cases for recursion are:

- $t_i^{n+1} = t_i^n$. This occurs when considering the probability of finding 0 faults and there is no further interference to include. In this case, t_i^n represents a possible value for R_i with a known probability. The pair $\langle t_i^n, \mathrm{p}(t_i^n) \rangle$ is recorded.

- $\mathrm{p}(t_i^n) < \rho$. A path is so unlikely to occur that it is insignificant, in which case this path may be ignored.

- $t_i^{n+1} > T_i - J_i$. Any analysis based on equation (4.5) cannot handle the possibility that the response time is greater than the period of a message. Therefore, when this occurs, the probability that the message is unschedulable must be recorded.

4.4 Interpretation of the Distribution

The immediate result of the analysis is a set of pairs $R_i = \langle t_i, \mathrm{p}(t_i) \rangle$. More usefully, a cumulative probability distribution can be plotted.

The nature of the analysis means that the expected results include some very small probabilities down to ρ, as well as larger values; all values are of interest. A linear scale would not show the smaller probabilities clearly enough, so a logarithmic scale is appropriate (indeed, on a linear scale, a graph shows very little of interest). However, in order to plot a cumulative distribution, the graph must display $(1 - \mathit{cumulative\ probability})$ to ensure that the small changes in probability are close to 0 rather than close to 1. The values on this axis can be interpreted as an upper bound on the probability of the response time exceeding the corresponding value on the horizontal axis.

Unfortunately, these transformations can be difficult to visualise, leaving a somewhat counter-intuitive plot. Care must be taken when interpreting the graph as it represents discrete data, not a continuous function. The correct way to join the points is as shown in Figure 4.4 so that the resulting line represents an upper bound on the results. By means of an illustrative example in Figure 4.4, a typical output of the analysis is shown.

With reference to the graph, one can infer for example that:

- $\mathrm{p}(t \geq 10) \leq 0.1$
  The analysis records that $\mathrm{p}(t \geq 10) = 0.1$, but the analysis is pessimistic, therefore the probability of a real frame having a response time longer than 10 is less than or equal to 0.1.

- $\mathrm{p}(t \geq 19) \leq 0.1$

- $\mathrm{p}(t < 10) > 0.9$

- The probability of a given message being delayed beyond the deadline is less than or equal to 0.01. This is equivalent to Navet's worst case deadline failure probability (WCDFP).

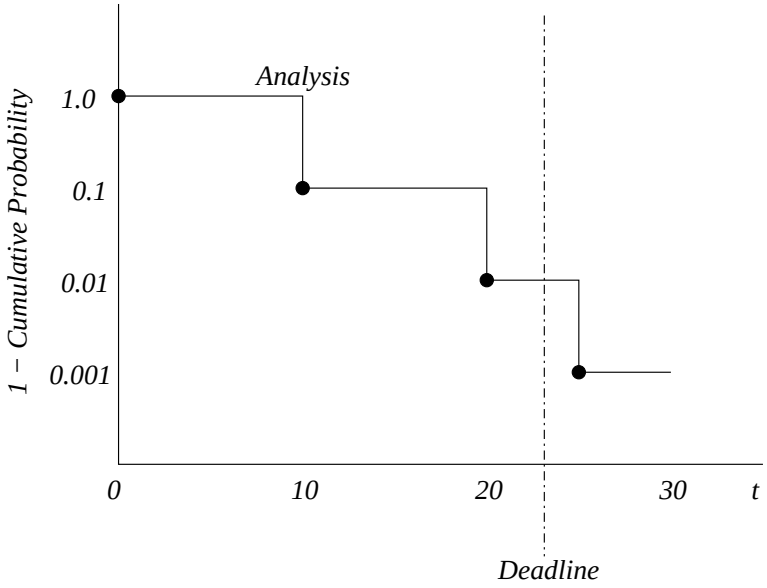


Figure 4.4: Interpretation of Results.

- If data obtained from extended monitoring of a real system or by simulation is plotted on the same axes, then if the analysis is pessimistic, the real data should always appear further to the left of the graph than the analysis line.

4.5 Probabilities, Complexity and ρ

The parameter ρ is an important parameter in the algorithm, relating accuracy, complexity and completeness. The effects of ρ are explored in this section.

4.5.1 Coverage

The algorithm necessarily cuts improbable branches of the tree using ρ. However, if there is a large number of branches that have probabilities of just below ρ then their summed probability can be significant, leading to probability tree whose total probability falls short of 1.

Therefore to assess the *completeness* of the search, the *coverage* can be defined as

the sum of the probabilities in all explored branches:

$$Q_i = p(N) + \sum_{\forall \langle t,p \rangle \in R_i} p \tag{4.10}$$

where $p(N)$ is the probability of being unschedulable, recorded by the analysis when $t_i^{n+1} > T_i - J_i$.

To be safe, any branches that are not covered should be treated as unschedulable. Therefore, a strict WCDFP can be found directly from the probability of being less than the deadline.

$$p(R_i > D_i) = 1 - \left(\sum_{\forall \langle t,p \rangle \in R_i | t \leq D_i} p \right) \tag{4.11}$$

To increase coverage, it is clear that ρ should be very small, however this naturally leads to an increase in complexity.

4.5.2 Complexity

The tree has the potential to grow exponentially, which would lead to infeasible computation times. However the algorithm keeps execution under control by capping many branches using ρ. Further, in general, the search tree becomes very skewed (as in Figure 4.3), which vastly reduces the overall size of the tree.

Larger values of ρ will result in shorter execution times: far fewer branches are explored because both the depth of the tree is reduced and there are fewer branches at each level. The parameter ρ is very important in controlling the growth of the tree.

As an example of the complexity, Section 4.9 presents results based on two case studies; an implementation of the analysis on a modern PC explored all the trees of the message sets in a few seconds. The number of branches explored by the analysis ranged from around 500 for the highest priority messages to just over 2,700,000 for the lowest priority message. The maximum recursion depth was 26. Infeasible computation times have not been observed for any message set tested so far with reasonable values of ρ.

Despite the potential for exponential growth, the analysis is much faster to compute than simulation or monitoring of the bus. This is because simulation would have to be performed for extended periods in order to be likely to observe infrequent scenarios, whereas the analysis systematically detects unlikely scenarios during the computation of the tree.

4.5.3 Setting ρ

As the previous discussion showed, coverage comes only at the cost of execution time, with ρ being the controlling parameter of this conflict. A value of ρ too small will result in infeasible execution times for the algorithm, larger values will lose coverage of the search space and therefore lose accuracy for low probabilities.

Setting ρ can be done on a message by message basis as each stream is considered individually.

Values of ρ between 10^{-10} and 10^{-20} are useful, depending on the message set and system requirements. This is a very large range. However, it is useful to note that there is little to gain by small changes in ρ, it may be practical to restrict ρ to values of the form 10^{-n}, where $n \in \{10..20\}$. Therefore trial and error is a valid means of setting ρ.

However, this may be done more formally by considering the failure requirements as follows. The resultant probability distribution refers to one single invocation of the message. It is common for a dependable system to have reliability requirements in the form "fewer than 10^{-9} failures per hour". If it is assumed that timing failures are independent, then it is easy to calculate the failure probability requirement *per invocation*. For example, for a periodic message with $T = 100\,\text{ms}$, there are 36,000 invocations per hour requiring a probability of failure per invocation of $10^{-9}/36,000 = 2.7 \cdot 10^{-15}$. To ensure precision, a value of ρ must be chosen such that it is an order of magnitude lower than the probability of failure per invocation. In this case a suitable value is $\rho = 10^{-16}$. This value can be adjusted if necessary if the coverage is deemed to be insufficient, or if the algorithm execution time is too long for a particular message set.

4.6 Implementation of the Probabilistic Algorithm

The analysis can be efficiently implemented as a recursive procedure, exploring the tree depth-first and using a simple data-structure to record the results. An algorithm is shown in Figure 4.5.

```
procedure Response(t, Δt, E_prev, p_path) is
   -- Recursive procedure to solve eqn (4.8)
   -- E_prev is E_i^{n-1} in eqn (4.9)
  if Δt = 0 then  -- Base case: Converged
     AddToPDF(t, p_path);
  else if t > T_i − J_i then  -- Base case: only R ≤ T
     AddToPDF(NOTSCHED, p_path);
  else  -- Recursive case
     for j ∈ {n ∈ ℕ | p_path·Poisson(n, Δt, λ) ≥ ρ} do
        p_j ←Poisson(j, Δt, λ);
        -- Probability of j errors
        E_j ←Errors(i, j, Δt);
        -- Overhead of j errors
        t_new ← C_i + B(i) + I(i,t) + E_prev + E_j;
        Response(t_new, t_new − t, E_prev + E_j, p_path · p_j);
     end for;
  end if;
end Response;;

for i ∈ messages do
  ClearPDF();
  Response(C_i, C_i, 0, 1.0);
  PrintPDF();
end for;
```

Figure 4.5: Analysis Algorithm.

The algorithm is presented in a form designed for clarity, rather than implementational efficiency. However, it should be noted that the test for the second base case ($\mathrm{p}(path) < \rho$) is implicit in the **for** loop. A real implementation of the algorithm should also make a number of further optimisations (such as calculating Interference and Blocking outside the loop). Also, extreme care should be taken if implementing with floating point arithmetic because the algorithm may need to manipulate both

large and very small values together while avoiding loss of precision. Functions such as Poisson() are particularly vulnerable. A high precision numeric library or calculations package is suggested.

4.7 Improving the Analysis

The algorithm in Figure 4.5 in Section 4.3 explores the search space generated by the worst case response time equations. The analysis is very general and can be applied in a number of scheduling scenarios including CPU scheduling. However, in the specific case of faults on CAN, it is possible to modify the analysis to reduce the computation time of the algorithm and eliminate branch pruning to achieve complete coverage.

Section 4.7.1 first makes a small improvement by pre-computing response times, then Section 4.7.3 exploits the specific fault model to produce a scheme which has a very low computation cost and achieves full coverage.

4.7.1 Pre-computing Response Times

As the examples later in this chapter illustrate, the shape of the probability distribution output is ‘stepped’. The cause is the simple nature of the error overhead function, equation (4.4). It is noted, therefore, that there are only a relatively small number of possible worst case response times. A large number of different scenarios contribute to the probability of each response time value; the probability of each response time is the sum of the probabilities of these scenarios.

Therefore, it is possible to pre-compute the set of possible response times up to some point (such as the period of the message, which is the limit of the analysis) and then calculate the possible scenarios which contribute to each response time. There is a computational advantage in doing this because:

- the size of the computation is reduced because several iterations of the worst case response time equations are able to be compressed into one iteration;

- computations involving *time* (worst case response times, interference *etc.*) are only done once, rather than throughout the algorithm.

Pre-computing the response times is done in the expected manner, by forming a recurrence relation from equation (4.12).

$$R_{i|K} = B_i + C_i + I(R_{i|K}) + E_{i|K}(R_{i|K}) \tag{4.12}$$

where

$$E_{i|K}(t) = K(E + \max_{\forall j \in \mathsf{hep}(i)} C_j) \tag{4.13}$$

4.7.2 Scenarios

After pre-computing the possible worst case response times, it is necessary to consider the scenarios that contribute to each possible value. In the next section, an analysis scheme is presented which avoids having to enumerate all possible scenarios, but the discussion of these scenarios in this section is useful to aid understanding of the scheme.

As Section 4.3.3 explained, the possible response times generate a set of non-overlapping intervals, as shown in Figure 4.6. The notation $e(K)$ is used to denote the number of faults that occur in time interval $(R_{i|K-1}, R_{i|K}]$ (or $(0, R_{i|K}]$ where $K = 0$).

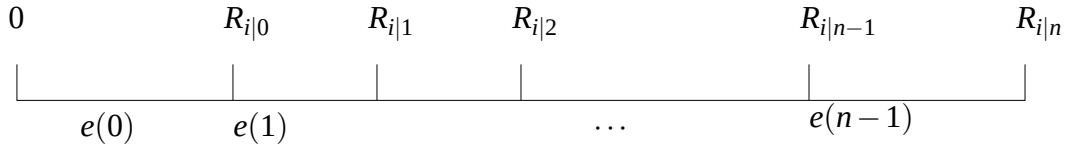


Figure 4.6: Response Times for Improved Algorithm.

Using the shorthand, $[210]$ to mean the scenario e(0)=2, e(1)=1, e(2)=0, Table 4.3 shows the scenarios which contribute to a given response time. Note that (for example) the sequence $[1020]$ cannot contribute to R_3, since the sequence begins $[10]$ which

Table 4.3: Enumeration of Scenarios.

Response Time	Possible Scenarios (Shorthand)	Number of Scenarios
$R_{i\|0}$	[0]	1
$R_{i\|1}$	[10]	1
$R_{i\|2}$	[200], [110]	2
$R_{i\|3}$	[3000], [2100], [2010], [1200], [1110]	5
$R_{i\|4}$	[40000], [31000], [30100], [30010], [21100], [21010], ...	14

contributes only to $R_{i|1}$ because at time $R_{i|1}$, there has been only one fault therefore the iteration of the WCRT equation terminates. This further illustrates the difference between this analysis and the previous WCDFP analysis which would have pessimistically considered [1020] to contribute to the response time for 3 faults, $R_{i|3}$.

The sequence of the number of scenarios which constitute each response time grows rapidly. It begins 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, and is known as the Catalan Series [150] given by the formula:

$$\frac{(2K)!}{K!(K+1)!} \tag{4.14}$$

where K is the number of faults, as in $R_{i|K}$.

Although the next section shows that is not necessary to enumerate all the scenarios, it is nevertheless possible to generate all scenarios described previously. For small values of K, complete coverage may be obtained, however for larger values of K, is it infeasible to enumerate all scenarios so branch pruning could be used.

Generating the scenarios could be done either by considering the sequence directly, or by considering a tree very similar to that used to explore the original analysis in Section 4.3. The only difference is that instead of each node representing an iteration of the worst case response time equation, in this case each node is a possible number of faults $e(x)$ where x is the depth of the tree. A further advantage of using this same tree-searching style of algorithm is that it is easy to incorporate branch pruning using ρ because this is simply a base case of the recursion.

One problem of using an analysis based on pre-computed worst case response times (rather than the general form of the analysis that was initially presented) is that it leads to a difficulty in determining the coverage. Response times greater than the deadline are indistinguishable from parts of the search tree that are not covered. Therefore, it is not possible to check the coverage directly. However, in general, since uncovered regions are added to the probability of being unschedulable, this does not affect the overall results of the analysis.

4.7.3 Efficient Probabilistic Analysis

From the precomputed response times, an efficient probabilistic analysis can be used to find the probabilities of the response times without enumerating all the scenarios. The technique is presented in this section.

Notation

For clarity, the notation $\mathrm{p}(n,t)$ is used to denote the probability of n faults occurring in interval t, $\mathrm{p}(n,t) \equiv \mathrm{p}_t(n)$, as defined in equation (4.2). The notation $\mathrm{p}(R_{i|K})$ is the upper bound on the probability that a frame i is affected by exactly K faults and hence may arrive no later than $R_{i|K}$. $R_{i|K}$ is precomputed as shown in Section 4.7.1.

The analysis will be derived by considering the scenarios which contribute to each particular response time.

Calculating $R_{i|0}$

The worst case response time with no faults, $R_{i|0}$, is the upper bound on the probability of faults not causing a frame i to exceed this time, $\mathrm{p}(R_{i|0})$. It is simply the probability that there are no faults in the interval $(0, R_{i|0}]$. As Table 4.3 showed, this is the only possible scenario that can produce a response time of $R_{i|0}$. This is exactly the same as the previous analysis.

$$\mathrm{p}(R_{i|0}) = \mathrm{p}(0, R_{i|0})$$

Calculating $R_{i|1}$

For the response time $R_{i|1}$, *i.e.* 1 fault, there is only one scenario which can cause this. Table 4.3 shows this to be $[10]$, there must be exactly one fault in $(0, R_{i|0}]$ and no faults in $(R_{i|0}, R_{i|1}]$. The previous section suggested that the probability of $[10]$ may be calculated by summing the probabilities of the scenarios (just one in this case).

However, an alternative approach is to begin with the probability of having exactly one fault in the interval $(0, R_{i|1}]$, which is $\mathrm{p}(1, R_{i|1})$. This can occur in only two ways: $[01]$ or $[10]$, of which only $[10]$ is of interest. The probability of scenario $[01]$ is already partially calculated because this is $\mathrm{p}(R_{i|0})$ multiplied by the probability of 1 fault in $(R_{i|0}, R_{i|1}]$.

$$\begin{aligned}\mathrm{p}(1, R_{i|1}) = & \\ & \mathrm{p}(R_{i|1}) && [10] \\ & + \mathrm{p}(R_{i|0})\,\mathrm{p}(1, R_{i|1} - R_{i|0}) && [01]\end{aligned}$$

Hence:

$$\mathrm{p}(R_{i|1}) = \mathrm{p}(1, R_{i|1}) - \mathrm{p}(R_{i|0})\,\mathrm{p}(1, R_{i|1} - R_{i|0})$$

Calculating $R_{i|n}$

Likewise, to calculate $\mathrm{p}(R_{i|2})$, is it possible to begin with the probability that there must be exactly two faults in $(2, R_{i|2}]$ and then exclude the scenarios where there were exactly 0 faults in $(0, R_{i|0}]$, or exactly 1 fault in $(0, R_{i|1}]$ since these scenarios would give rise to smaller response times.

$$\begin{aligned}\mathrm{p}(R_{i|2}) = & \mathrm{p}(2, R_{i|2}) \\ & - \mathrm{p}(R_{i|1})\,\mathrm{p}(1, R_{i|2} - R_{i|1}) \\ & - \mathrm{p}(R_{i,|0})\,\mathrm{p}(2, R_{i|2} - R_{i|0})\end{aligned}$$

The result is generalised as follows. The probability of exactly n faults in $R_{i|n}$ is derived directly from the Poisson distribution equation, $\mathrm{p}(n, R_{i|n})$. However only some permutations of faults can possibly lead to such a response time. The permutations

which cannot lead to $R_{i|n}$ are those which would lead to a response time $R_{i|j}$ where $j < n$.

If there are j faults in $(0, R_{i|j}]$ then (because there are n faults in $(0, R_{i|n}]$) there must be $n-j$ faults in $(R_{i|j}, R_{i|n}]$. So, the probability of j faults in $(0, R_{i|j}]$ given that there are n faults in $(0, R_{i|n}]$ is $\mathrm{p}(R_{i|j})\,\mathrm{p}(n-j, R_{i|n} - R_{i|j})$. This value can then be subtracted from the probability $\mathrm{p}(R_{i|n})$.

The resulting general equation for the upper bound on the probability of worst case response time $R_{i|n}$ is:

$$\begin{aligned} \mathrm{p}(R_{i|n}) = & \mathrm{p}(n, R_{i|n}) \\ & - \sum_{j=0}^{n-1} \mathrm{p}(R_{i|j})\,\mathrm{p}(n-j, R_{i|n} - R_{i|j}) \end{aligned} \tag{4.15}$$

Results obtained from this scheme should also have greater accuracy than the tree based probabilistic schemes because there are far fewer calculations, hence less accumulated rounding errors. Additionally, it should be noted that there is no branch pruning; therefore full coverage should be achieved.

Implementation of this is trivial, so code is not shown. A software implementation based on equations (4.12) and (4.15) was used to perform the study in Section 4.9.

4.8 Multiple Invocation Analysis

The efficient analysis explained in Section 4.7.3 is applied only to the critical instant where all frames are released. As Chapter 3 showed, there are advantages to analysing other invocations in the hyperperiod rather than just the worst case. A probabilistic analysis of all invocations in the hyperperiod is the next logical extension. However, there are prohibitive difficulties in doing so and no probabilistic multiple invocation analysis is presented in this thesis.

It is possible to construct an implementation of such an analysis, based on the original tree-searching analysis from Section 4.3 with trivial modifications to consider multiple invocations. Experiments performed with the tool showed that the expected

advantages of considering multiple invocations can be achieved within the probabilistic framework. However, the computation required to perform the analysis for realistic length hyperperiods is prohibitive; whilst for a single invocation, the tree is small enough to be practical, for multiple invocations the tree grows infeasibly large.

A multiple invocation analysis based on the later, more efficient approach in Section 4.7.3 does not follow trivially because of difficulties calculating the idle time in a probabilistic manner. The problem arises because the approach requires precalculation of all possible response times, yet in the multiple invocation analysis, there are potentially many thousands of response times. The response time of each invocation depends on the response times of the previous invocation (because of the idle time calculation). Therefore, immediately there are very many possible response times.

Perhaps one method to achieve a probabilistic multiple invocation analysis is to strive only for a pessimistic upper bound, hence a single valued idle time may be used for the calculations of each iteration. This may be considered in future work.

Despite the lack of a probabilistic multiple invocation analysis, its use is not necessary to gain understanding of the probability of failure for an event-triggered bus. The single invocation probabilistic analysis allows detailed consideration of the worst case, which is the most useful case to consider because it forms an upper bound on all other cases.

4.9 Evaluating the Analysis

In this section, the new probabilistic analysis is evaluated by use of two case studies. The results of the analysis are compared to data obtained through software simulation, and to a previous study.

4.9.1 Peugeot Message Set

The first case study used was proposed by Navet [114]. It is used in this thesis in order to enable a comparative study with his previous result. The original source was provided by Peugeot-Citroen Automobiles Company. There are 12 periodic messages

Table 4.4: Peugeot Example Message Set.

Priority i	**Length** C_i	**Period** T_i(ms)	**WCRT** R_i(ms)
12	528	10000	1028
11	328	14000	1368
10	328	20000	1708
9	288	15000	2008
8	408	20000	2428
7	408	40000	2848
6	368	15000	3228
5	408	50000	3648
4	368	20000	4028
3	488	100000	4448
2	408	50000	4708
1	248	100000	4720

from six devices in a prototype car. A data-rate of 250 kbit/s is used. The message set is not a particularly 'strenuous' one: under no-fault conditions, the worst case response time for all messages is less than the shortest period in the set. Therefore, no frame can experience interference from any other frame more than once. The bus utilisation is only 21.6%. It should be noted that the messages are also schedulable at the slower (and therefore less prone to faults[3]) data-rate of 125 kbit/s, and that the priorities are *not* ordered rate monotonically (RMPO). For all messages, the deadline is equal to the period and there is no release jitter. The message set is shown in Table 4.4 with standard worst case response time calculations shown for comparison.

The analysis in Section 4.7.3 was applied at the fault rate of $\lambda = 30$ faults per second. This value of λ is used because it has been frequently used in the past [114, 127] as an expected number of faults in an aggressive environment. Reading the results directly from Table 4.5 one can get a feel for the analysis. Only three frames are shown, however they are representative of the other frames: they all show similar results. It can be seen that the probabilities of messages being significantly delayed are very low. The probability of any message being delayed beyond its deadline is insignificant (less than ρ, see Section 4.5). The full data set is plotted as a cumulative probability graph in Figure 4.7. The 'steps' are due to the nature of the error overhead function, $E_i(t)$,

[3] §2.5.3.

Table 4.5: Probabilistic Analysis Results, (Peugeot Set, $\lambda = 30.0$).

(a) Frame 12

R_i (ms)	$\mathrm{p}(R_i)$
1028	$9.696307 \cdot 10^{-01}$
1684	$2.932066 \cdot 10^{-02}$
2340	$1.009100 \cdot 10^{-03}$
2996	$3.795376 \cdot 10^{-05}$
3652	$1.514530 \cdot 10^{-06}$
4308	$6.300757 \cdot 10^{-08}$
4964	$2.703161 \cdot 10^{-09}$
5620	$1.187428 \cdot 10^{-10}$
6276	$5.314400 \cdot 10^{-12}$
6932	$2.414760 \cdot 10^{-13}$
7588	$1.111030 \cdot 10^{-14}$
8244	$5.165844 \cdot 10^{-16}$

(b) Frame 5

R_i (ms)	$\mathrm{p}(R_i)$
3648	$8.963359 \cdot 10^{-01}$
4304	$9.618337 \cdot 10^{-02}$
4960	$7.016588 \cdot 10^{-03}$
5616	$4.374734 \cdot 10^{-04}$
6272	$2.516691 \cdot 10^{-05}$
6928	$1.382444 \cdot 10^{-06}$
7584	$7.381265 \cdot 10^{-08}$
8240	$3.869713 \cdot 10^{-09}$
8896	$2.004302 \cdot 10^{-10}$
9552	$1.029642 \cdot 10^{-11}$
10208	$5.259833 \cdot 10^{-13}$
11404	$2.633599 \cdot 10^{-14}$
12060	$1.754993 \cdot 10^{-15}$

(c) Frame 1

R_i (ms)	$\mathrm{p}(R_i)$
4720	$8.679684 \cdot 10^{-01}$
5376	$1.205092 \cdot 10^{-01}$
6032	$1.069119 \cdot 10^{-02}$
6688	$7.773385 \cdot 10^{-04}$
7344	$5.062092 \cdot 10^{-05}$
8000	$3.079614 \cdot 10^{-06}$
8656	$1.791207 \cdot 10^{-07}$
9312	$1.009935 \cdot 10^{-08}$
9968	$5.568988 \cdot 10^{-10}$
11164	$2.972493 \cdot 10^{-11}$
11820	$2.065001 \cdot 10^{-12}$
12476	$1.227213 \cdot 10^{-13}$
13132	$6.917263 \cdot 10^{-15}$

as explained in Section 4.7.

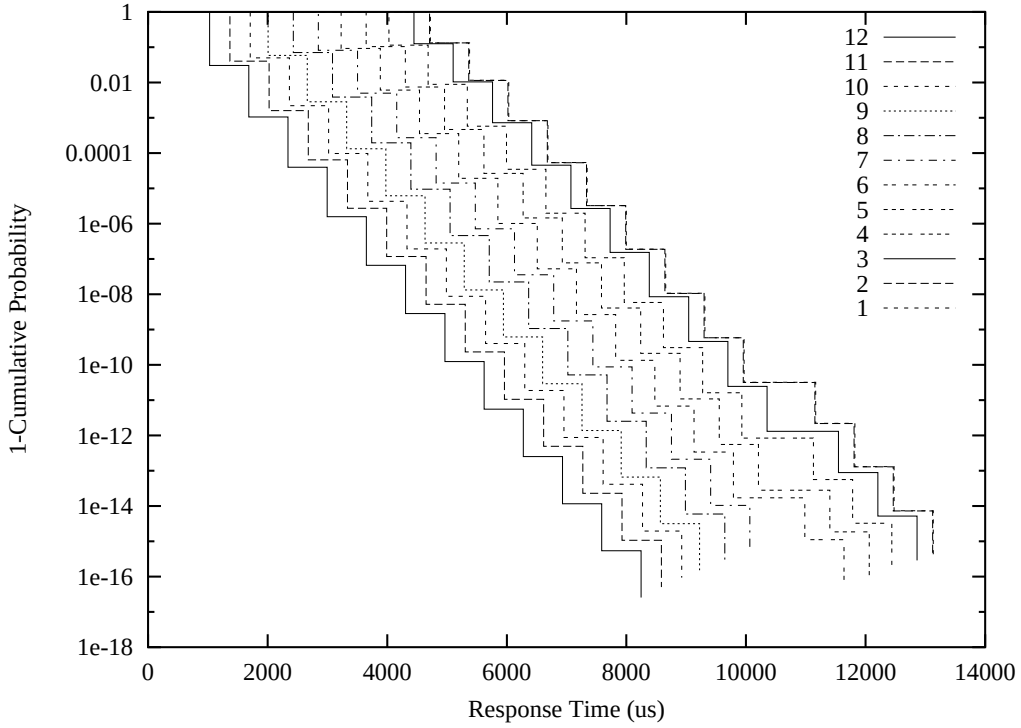


Figure 4.7: Probability Distribution of Response Times for All Frames (Peugeot Set, $\lambda = 30$).

The software simulator noted in the previous chapter is used to accurately model the CAN bus in the presence of faults. It is possible to simulate for extended periods, or to repeatedly simulate conditions after a critical instant.

In order to explore the pessimism and accuracy of the analysis, the analysis results for the frame with priority 5 are compared to the repeated simulation of a critical instant. The same Poisson fault model of $\lambda = 30$ faults per second was used for both the analysis and simulation, and the simulation was run 1,500,000 times. The results of both the simulation and the analysis for this frame are plotted in Figure 4.8. Note that both the simulation and analysis refer only to conditions following a critical instant—other invocations are not considered here.

The analysis and the simulation results are very similar. We consider first the response times with probability greater than about 10^{-6} (near the top of the graph). The analysis is slightly pessimistic: shown by the fact that the analysis line is always to

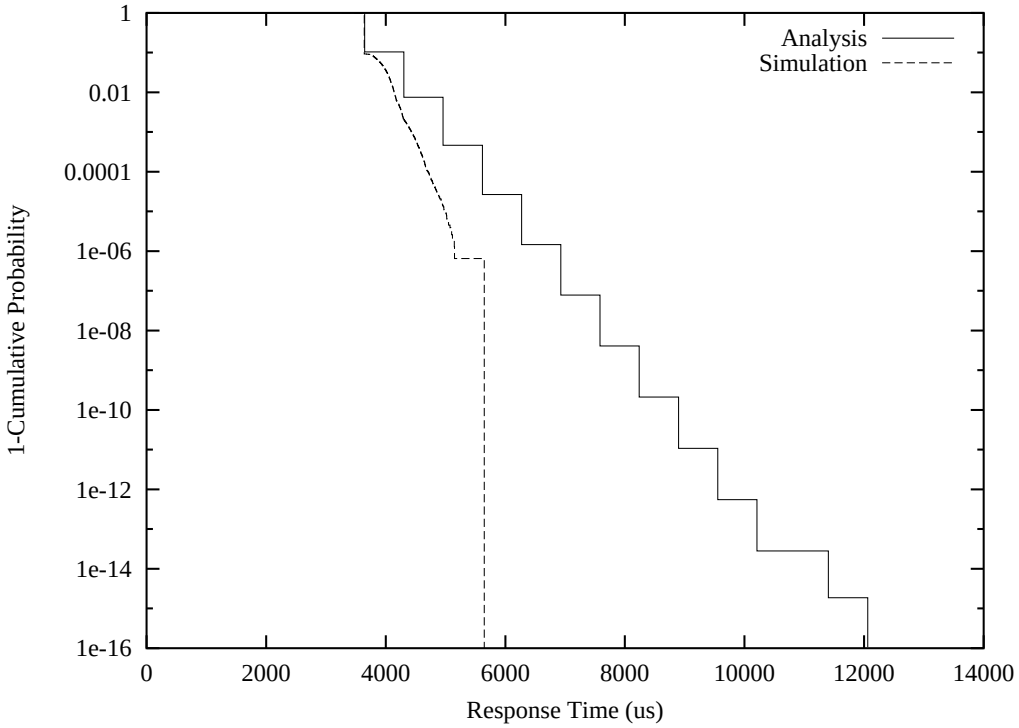


Figure 4.8: Response Time Probability, Analysis vs. Simulation, Frame 5, (Peugeot Set, $\lambda = 30$).

the right of the simulation line. The pessimism here comes mainly from the fact that the analysis always uses a maximal error function $E_i(t)$ whereas the simulation more accurately models network faults occurring in frames. However, the pessimism is not extreme—as can be see from the graph, the lines are very close.

For the longer response times at low priorities (lower in the graph), the simulation and analysis results differ. The longer response times suggested by the analysis never occurred in the simulation; they are unlikely to because the simulation run had far fewer than 10^{13} iterations. Yet the analysis predicts that such response times may occur during longer periods of execution. It is these low probabilities which need to be carefully considered in a critical system.

The advantage of probabilistic analysis over simulation or monitoring is clear: in order to be likely to observe events at probabilities as low as 10^{-13}, it would be necessary to simulate for at approximately the reciprocal of this number of times [94] which is clearly infeasible. Based on the simulator used here, this would take approximately

80 years of simulation, whereas the analysis took only seconds to achieve the result.

In comparison with Navet [114], the analysis in this thesis is far less pessimistic. Navet shows the worst case deadline failure probability for frame 5, $p(R_i > 50ms)$, with $\lambda = 30$ to be approximately 0.045. According to the analysis presented in this chapter, the probability of any frame being unschedulable is insignificant. As the deadline was never approached in the 1,500,000 simulation runs, this shows that 0.045 is indeed an overly pessimistic value.[4]

4.9.2 Probabilistic Evaluation of SAE Benchmark

The second case study used to evaluate the analysis is Tindell's widely published simplification [159] of the Society of Automotive Engineers (SAE) Benchmark [147], see Table 3.1 on page 91. Release jitter is neglected for the following experiments. Recall that the message set is only just schedulable, frames 12, 9 and 8 are particularly vulnerable to missing their deadlines because their worst case response times are close to their deadlines.

The analysis was performed with $\lambda = 10$ faults per second, a lower value than used in the previous Peugeot tests since the message set is a more demanding one; 10 faults per second has also been used as an expected fault model on CAN previously [127]. The results of the analysis are similar in form to the previous case study, and are plotted in Figure 4.9.

Notice that not all the lines in Figure 4.9 continue down to the very low probabilities, frame 15 is an example of this.The probability at the lowest point on a line can be interpreted as the probability of the message being unschedulable, taking coverage into account. As the efficient analysis is used, it achieves full coverage.

However, if the tree-based analysis is used then the lowest point of the line is determined by a combination of two factors: either because the analysis does not consider response times greater than the period of the message; or the coverage is low.

[4] The fault model used is similar to Navet's (Poisson, same λ) so it is appropriate to compare these values. Nevertheless, the models *are* different (no burst errors) so a direct comparison should not be misinterpreted. See Section 4.2.2 for further details of the pessimism.

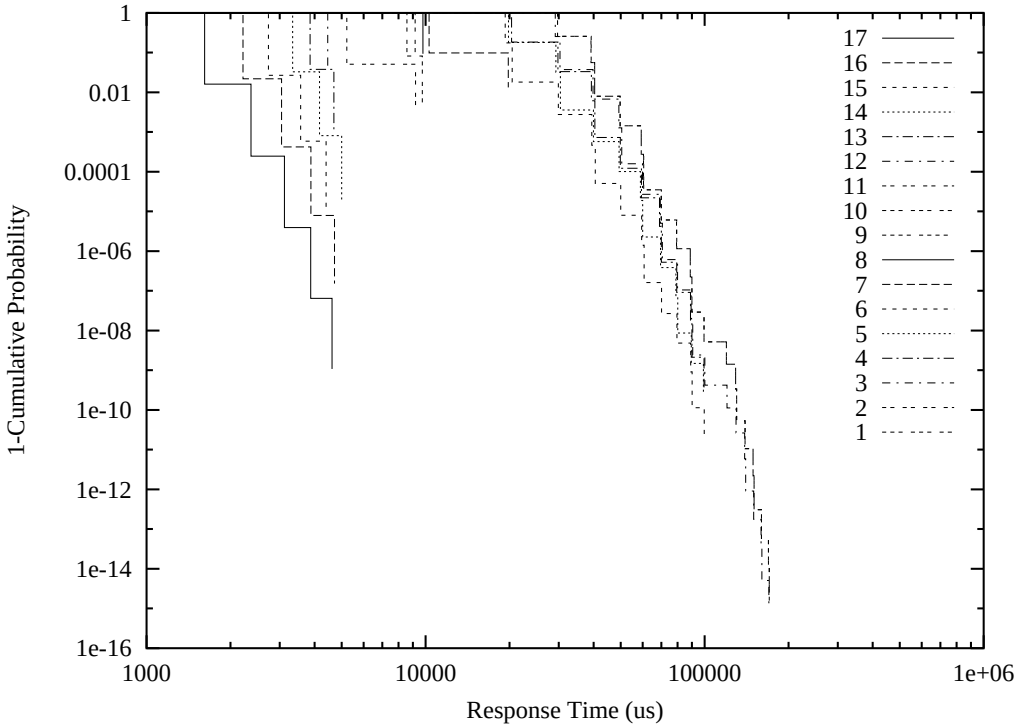


Figure 4.9: Analysis of all frames (SAE benchmark, $\lambda = 10$).

High-mid Priority Frames

The high and mid-priority frames in Figure 4.9 (except the very highest priority message) have response times very close to their periods and so have relatively high probabilities of deadline failure, *e.g.* for message 12, $p(R_{12} > D_{12}) = 0.0416$. Therefore (depending on the particular requirements of the system) the analysis here might indicate that the probability of deadline failure at a critical instant with this message set is not acceptable.

Low Priority Frames

For the lowest priority message in this example, they have such long deadlines that the probability of network faults leading to a deadline failure is very low.

Message 15 in Detail

The analysis results for message 15 (which is the third line from the left in Figure 4.9) appear in Table 4.6 expressed to 4 significant figures and are plotted (also with the simulation results) in Figure 4.10. The WCDFP for one invocation is then calculated as $1.208 \cdot 10^{-5}$, this being an upper bound for the real probability of frame 15 missing a deadline.

Table 4.6: Probabilistic Analysis of Frame 15 (SAE set, $\lambda = 10$).

R_i (ms)	$p(R_i)$
2736	$9.730 \cdot 10^{-01}$
3568	$2.640 \cdot 10^{-02}$
4400	$5.760 \cdot 10^{-04}$

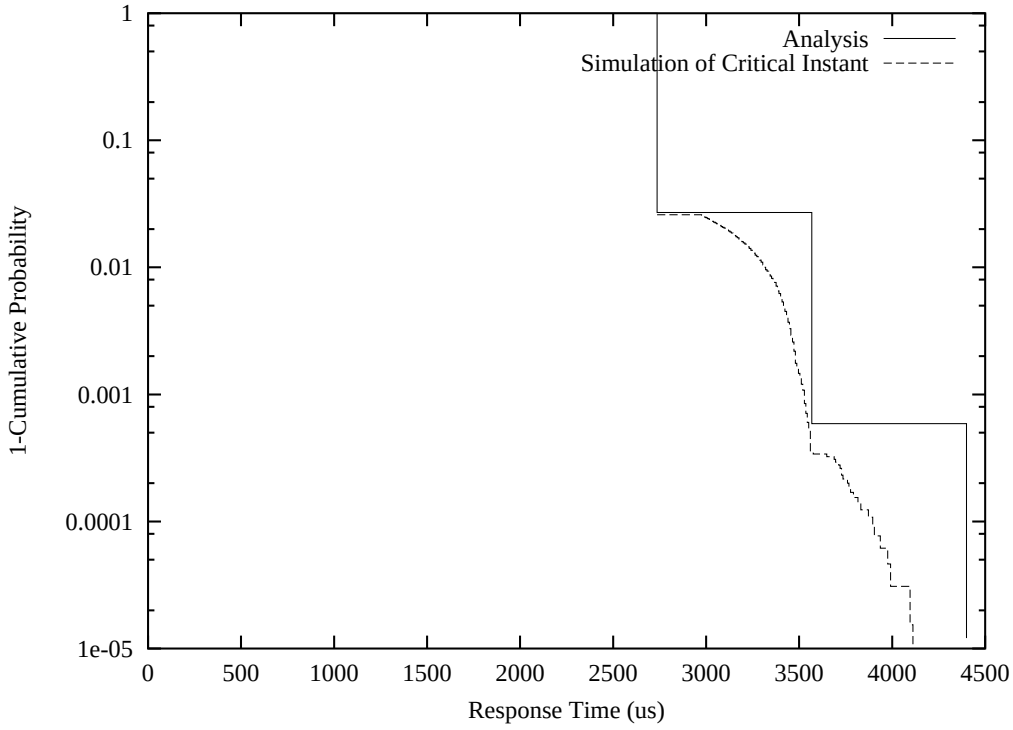


Figure 4.10: Analysis and Simulation for Frame 15 (SAE benchmark, $\lambda = 10$).

Again, the simulations confirm the analysis results and indicate that the analysis is slightly pessimistic in all cases.

Table 4.7: Comparison of new Analysis, Previous Approach and Simulation.

	Probability of Deadline Failure		
Priority	**New Analysis**	**Navet**	**Simulation**
17	$1.854660 \cdot 10^{-07}$	$3.072870 \cdot 10^{-07}$	0.000000
16	$9.368960 \cdot 10^{-06}$	$1.255530 \cdot 10^{-05}$	0.000000
15	$2.638460 \cdot 10^{-04}$	$3.034240 \cdot 10^{-04}$	0.000000
14	$4.031250 \cdot 10^{-04}$	$4.468920 \cdot 10^{-04}$	$9.000000 \cdot 10^{-06}$
13	$8.015490 \cdot 10^{-03}$	$8.290030 \cdot 10^{-03}$	$8.300000 \cdot 10^{-05}$
12	$1.198650 \cdot 10^{-01}$	$1.198650 \cdot 10^{-01}$	$2.029000 \cdot 10^{-03}$
11	$2.485930 \cdot 10^{-02}$	$3.055750 \cdot 10^{-02}$	$4.700000 \cdot 10^{-05}$
10	$3.338800 \cdot 10^{-02}$	$3.384190 \cdot 10^{-02}$	$3.780000 \cdot 10^{-04}$
9	$2.360710 \cdot 10^{-01}$	$2.360710 \cdot 10^{-01}$	$4.377000 \cdot 10^{-03}$
8	$2.496980 \cdot 10^{-01}$	$2.496980 \cdot 10^{-01}$	$3.916900 \cdot 10^{-02}$
7	$9.291990 \cdot 10^{-02}$	$1.176530 \cdot 10^{-01}$	0.000000
6	$4.822250 \cdot 10^{-06}$	$1.530720 \cdot 10^{-05}$	0.000000
5	$7.867910 \cdot 10^{-06}$	$1.614900 \cdot 10^{-05}$	0.000000
4	$2.880640 \cdot 10^{-05}$	$6.730180 \cdot 10^{-05}$	0.000000
3	$1.021330 \cdot 10^{-17}$	$2.838210 \cdot 10^{-17}$	0.000000
2	0.000000	$2.943790 \cdot 10^{-17}$	0.000000
1	0.000000	$2.865470 \cdot 10^{-17}$	0.000000

4.9.3 Comparison with Previous Approaches

The result of the new analysis is a probability distribution; the result of Navet's analysis is a single probability of failure. Therefore it is easy to compare the two analyses by considering a single point in the probability distribution (the deadline), as explained in Section 4.5.

Table 4.7 compares the results of applying the new probabilistic analysis at the deadline, Navet's analysis (after correcting the multiple faults problem, Section 4.2.3) and simulation of the SAE benchmark.

It is clear that the new analysis offers an improvement in pessimism over the old analysis; in some cases by a large amount. To further compare the results, the data is plotted, however due to the wide range of the probabilities, it is not useful to plot all messages on the same axes, even using a logarithmic scale. Therefore, the data plotted in Figure 4.11 is scaled by normalising against the results of the new probabilistic analysis. The vertical axis is therefore the ratio between the probability of deadline

failure of the new analysis and the other lines.

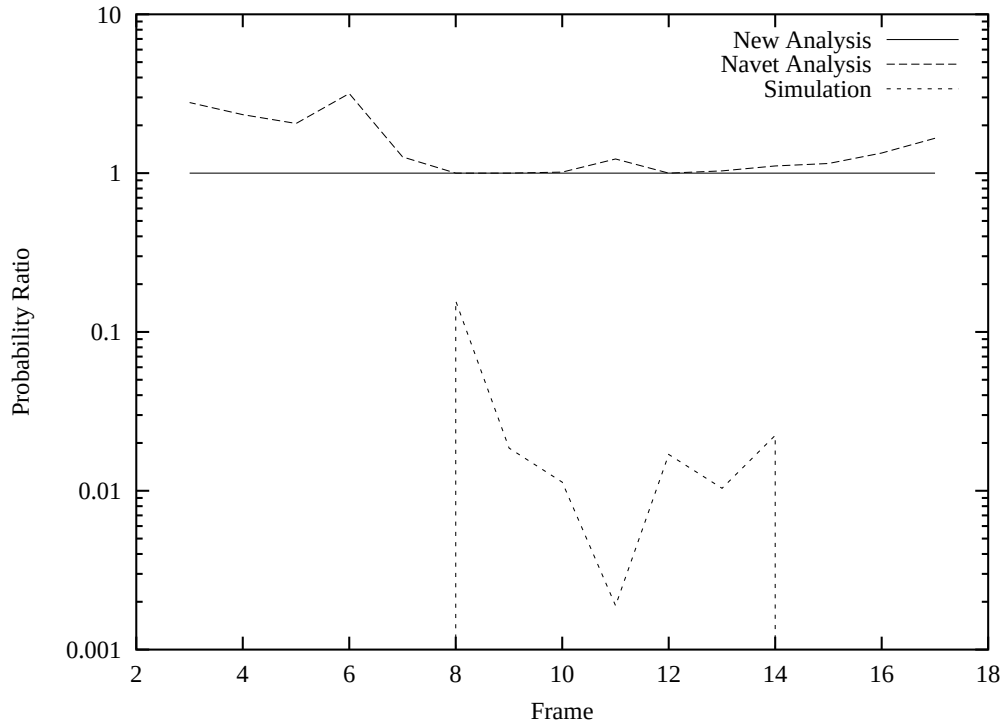


Figure 4.11: Comparison of Navet and New Probabilistic Analyses, Normalised Against New Analysis.

The main difference between the two analyses is that the one presented in this thesis removes the source of pessimism in Navet's analysis where faults after a message has arrived (but before the deadline) are assumed to delay the message. Therefore, it is possible to provide an approximate value (for this single test case) of the impact of this particular form of pessimism. For messages whose response time is very close to the deadline, there is little difference, indeed where one fault would exceed the deadline, the results are mathematically identical (*e.g.* frame 12). For other frames with a lower probability of failure (*e.g.* 6, 5, 4) the new analysis produces values approximately 30%–50% lower. For messages which are extremely unlikely to miss a deadline, (1, 2, 3—streams 1 and 2 not shown) it is meaningless to compare the results because: (a) the probabilities are *very* low and (b) the rounding errors introduced by the implementations are of the same order of magnitude as the results).

However, there is still a significant difference between the new analysis and the sim-

ulation results, this is due to other sources of pessimism in the analysis, as discussed in Section 4.3.1.

Other Differences in Application

In addition to the pessimism improvements, there are other differences between the probabilistic analysis introduced in this section and previous approaches.

One additional use of the new analysis is that instead of calculating the probability for a single point (*e.g.* deadline) the response time *distribution* may be used to answer the question "with an acceptable probability p, what is the longest response time expected?".

Further, this form of analysis can be used to allow a far greater understanding of the behaviour of a flexible communication system with faults. Rather than simply a 'Yes/No' schedulability test, when the results are plotted, the designer can view the data graphically, to visualise how the reliability of the system changes as the level of faults increases.

4.10 Summary

The flexibility of an event-based bus allows it to cope well with unpredictable faults by retransmitting corrupt frames as necessary. In this chapter, an argument for using a random fault model based on a simple Poisson arrival model was presented, accommodating the idea that faults may arrive at any time. Further, there is always a non-zero probability that faults will cause delays leading to a deadline failure. Hence it is usually necessary to write code that is sufficiently tolerant of occasional late or omitted messages.

The standard industrial practice of merely applying the standard schedulability analysis [159] is not sufficient to be able to *guarantee* performance in critical systems because it relies on the assumption of a minimum inter-arrival time between faults. This chapter has shown that no analysis can provide a guarantee of performance, but

instead, it is possible to gain a probabilistic understanding of the response times expected.

Analysing the workings of a flexible system is difficult; in general real-time research has focused only on worst case analysis. Applying probabilistic notions directly to worst case analysis [38] for CAN with faults leads to very pessimistic results, as Section 4.2.1 showed. Considering the bus in a more precise way improves the pessimism, but Section 4.2.2 showed that previous research still shows significant pessimism.

The main contribution of this chapter is a new analysis technique which eliminates the bounded fault model, and the requirement to derive a level of confidence in such a model. Instead the probabilistic fault model allows for the concept that faults can occur at any time, with no limit to their frequency.

The new analysis removed a pessimistic assumption in a previous form of analysis [114]. Specifically, the new analysis removes the assumption that a fault occurring after a frame is delivered may cause the frame to be delayed. In general, the new analysis produces accurate probabilities of failure which are significantly lower (less pessimistic) than were previously obtainable. Sources of pessimism are naturally still present, however, the remaining pessimism is not extreme; the analysis data closely matches results obtained by accurate simulation.

The analysis has been completely automated, execution times of the algorithm were no more than a few milliseconds for the typical message sets used. Further, no changes are required to the system in order to perform this analysis as it models a standard CAN bus. Tool support can be provided in a way that is compatible with standard practice tools which currently use Tindell's schedulability analysis.

The result is an analysis which is both practical to use and accurately models the flexibility of real-time event-triggered communication, adding to the evidence in Chapter 3 that the flexibility of an event-based bus can be exploited to provide predictable systems.

5 Predictable Failure

A FAULT-MODEL IS ESSENTIAL for any form of response time analysis, yet even the use of a non-bounded, probabilistic model in the previous chapter does not guarantee correct operation under high-levels of faults. In this chapter, a CAN based protocol called Timely-CAN (TCAN) is developed which *under fault conditions* sacrifices assured delivery (that messages will arrive through eventually) for timeliness (any messages that arrive will arrive on time). This leads to an effective scheme for providing predictable behaviour under failure conditions [33].[1]

5.1 Approach and Justification

Previous chapters have shown that missing deadlines in an analysable way is both acceptable and useful in a real-time system. Furthermore, it is clear that the unpredictable nature of faults means that it is not possible to ever *guarantee* both assured and timely delivery of messages (although as the previous chapters have shown, the number of deadlines missed is analysable and can be very small).

In the presence of faults, the CAN protocol attempts to provide *assured* delivery, which it can only achieve at the expense of *timeliness* of messages. In general, the aims of *assured* and *timely* delivery are in conflict. CAN sits at one end of the scale providing assured delivery with no attempt at timely delivery—the CAN protocol itself does not even assimilate the concept of timely delivery. At the other end of the scale are time-triggered protocols such as TTP and Time-triggered CAN (TTCAN) where frames are transmitted at predetermined times. These focus on timely delivery

[1] In the original conference publication, TCAN was named LST-CAN. The names are synonymous, but TCAN is preferred.

at the expense of assured delivery, since if a fault affects a frame then that frame is abandoned (no re-delivery attempt is made).

In real-time systems, it is widely accepted [39] that *time* should be treated as a first class entity, rather than something that is added on afterwards. This is opposite to CAN where, time is not directly considered. Yet, as previously indicated, a criticism of time-triggered protocols is that they may have a low reliability in the presence of network faults because they do not have the flexibility to provide efficient active fault tolerance. In this chapter a simple extension to CAN is presented, TCAN, which is placed somewhere between CAN and TTCAN. It does this by directly considering both timely and assured delivery, each where it is useful to do so.

As timely delivery and assured delivery are in conflict, the protocol provides a best-effort approach to assured and timely delivery on CAN where if it is possible to deliver a frame in a timely fashion, then the frame is delivered, otherwise the frame is not delivered at all. The bandwidth released by not transmitting frames that would arrive late can be used to absorb delays and hence help to provide timely delivery of other frames.

To justify this approach it is considered that a late message has no value according to the firm fault model advocated in earlier chapters and common in industrial practice.[2] As further justification for using the firm model: in a hard real-time system if a message does not arrive on time then it is considered a fault and error recovery operations are performed. Therefore, there is no use in attempting delivery of a frame after its deadline has expired. This behaviour ties closely with the timed asynchronous model of communication [46].

Additionally, since faults are considered to be unpredictable, even if analyses such as the schemes introduced earlier in this thesis are used, off-line *absolute guarantees* of timely behaviour are not possible. During the lifetime of the system, exceptional bursts of high levels of faults may occur, the effects of which are beyond the analytical scenarios considered off-line and so on-line schemes are required if predictable behaviour is to be maintained.

As Chapter 3 showed, by allowing a small number of deadlines to be missed (in an

[2] See Sections 2.4.2 and 2.1.5.

analysable way), very large improvements can be made on the quantity of data that a bus can safely carry. Now, improvements can be made flexibly at run-time, in addition to just considering the behaviour through off-line analysis.

5.2 Intuition

Chapter 3 showed how weakly-hard analysis can take advantage of the fact that in an event-triggered bus (such as CAN), frames rarely take as long as their worst case response time, R_i, to reach their destination. Most of the time, a frame will arrive considerably earlier [24, 21]. This is because the worst case scenarios upon which the worst case response time R_i is derived only occur infrequently. In particular,

- the amount of interference that each frame can possibly receive from other frames varies on each invocation of a frame;
- when other factors such as the maximum bit-stuffing assumption, blocking and jitter are not producing their maximal overhead, the response time of a particular frame is reduced.

Additionally from Chapter 4:

- faults affecting the system are not predictable and certainly not consistent, therefore infrequent, but bad, scenarios of faults would be expected to be observed.

Therefore, for much of the time, there is considerable slack in the system which may be devoted to error correction (retransmissions). However, when near-worst-case scenarios are observed, the available bandwidth is limited, therefore the desire to retransmit corrupted frames has to be balanced against the delaying effect that this may have on other frames.

The TCAN protocol presented in this chapter aims to exploit the flexibility of an event-triggered bus in a predictable way by only providing error recovery when it is useful to do so.

5.3 The Timely-CAN Protocol

In this section the details of the TCAN protocol are explained. In outline, the section proceeds by arguing that some frames need not be transmitted, then a policy for which frames to not transmit is derived. Finally, semantics for the transmitters and receivers are defined.

5.3.1 Aborts

In a critical system, it is usually necessary to consider the consequences of not receiving a message (for example if the sending process fails, or in the event of transient or permanent bus communication failures). It is usual to write code to handle losing a message safely. Indeed, Chapter 4 showed that this is often necessary.

This chapter continues with this argument and supposes that in the general case, losing a message should not be considered fatal and the system should continue to function as normally as possible without the message. Control laws, for example, are generally able to maintain stability and work effectively if a small number of data values are lost (vacant sampling) [116].

Considering the firm deadline model, advocated in earlier chapters (where there is no utility in delivering a late message), it is noted that CAN makes repeated attempts to transmit a message, whether it is late or not. Removing the requirement for assured delivery allows the use of error recovery techniques that make informed decisions about whether or not to transmit specific frames.

To maintain terminology with current hardware devices and literature, the term *abort* when applied to a CAN frame will mean to cease attempting to transmit a frame. This corresponds to removing a frame from a transmission queue in a typical CAN hardware device, rather than stopping frame transmission in mid-frame.

5.3.2 Example

The basis of the *TCAN* protocol is to *never transmit a message which would arrive late* (due to errors), in order to provide bandwidth to ‘absorb’ delays. An aborted message

can be considered to be a form of *real-time error confinement*.

The scheme may be explained by example: Figure 5.1 shows how the protocol can ensure the deadlines of other messages on the bus after a continuous block of interference. Figure 5.1(a) shows a scenario where frames are scheduled in priority order. The points marked **X** are deadlines set by some appropriate scheme (see later).

In Figure 5.1(b), a burst of network interference prevents any message from appearing correctly on the bus for some time, *all* messages are delayed and arrive after their deadlines. Figure 5.1(c) shows how the protocol suggested deals with errors: messages 1 and 2 cannot possibly arrive before **X**, therefore they are abandoned, which in this case allows all other messages to arrive on time.

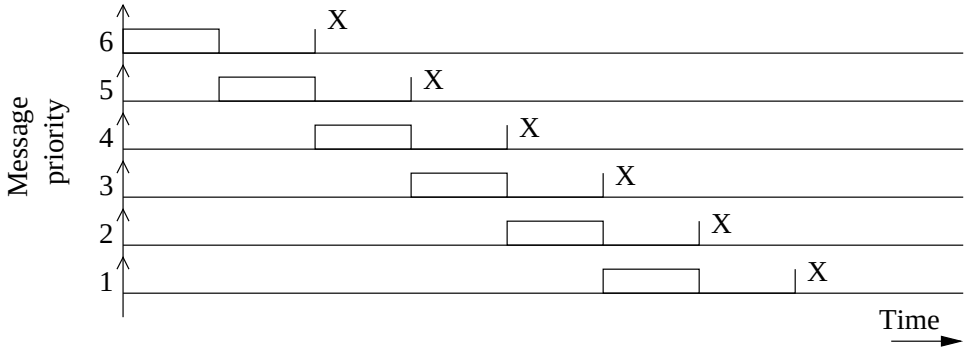
5.3.3 TCAN in detail

Time is used directly in the protocol: each frame is associated with a *delivery threshold* time which is known *a priori* to all nodes (*i.e.* common knowledge). The delivery threshold can be viewed as a deadline for transmission and it may be treated as a firm deadline, but note that this is not necessarily the same as the actual deadline which is derived from application requirements.

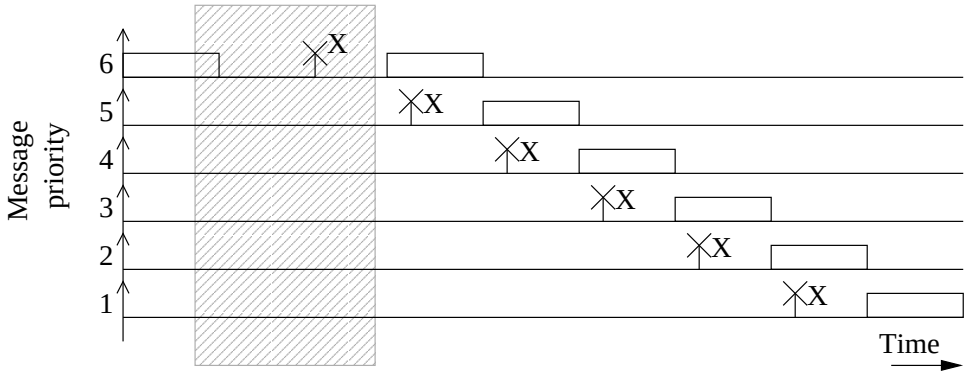
On the bus during a message transmission, there are two participating classes of node: one or more transmitters and zero or more receivers. Informally: the transmitters must ensure that a frame does not appear on the bus after the threshold, and the receivers must only listen for the frame until the threshold. The responsibilities of each are explained below in detail.

Initially, a single global clock is assumed with which all nodes are perfectly synchronised; this (unrealistic) requirement is relaxed later.

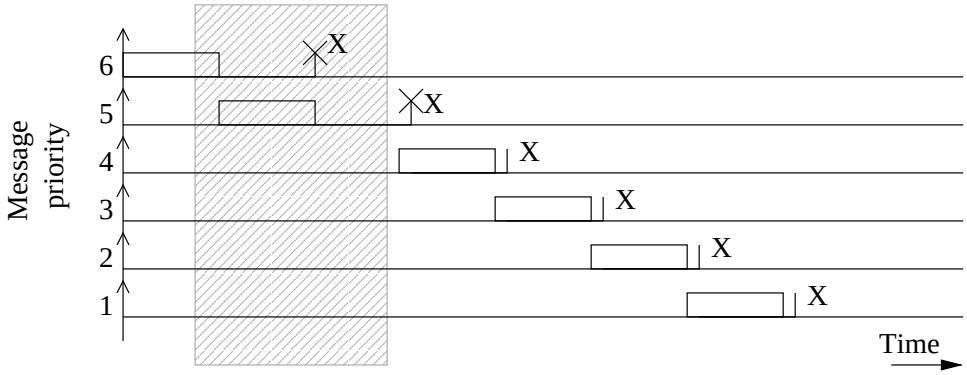
Using similar notation as used in Chapter 3, for each frame $\langle i,k \rangle$ (the kth message within a stream i) there is a globally known threshold, $X_{i,k}$. This is an absolute time with respect to the global clock. A globally known threshold is available with periodic messages, for example. Other messages where it is not possible to state a global delivery threshold are also considered in Section 5.6. There are a number of possible schemes for determining a suitable delivery threshold; these are discussed in



(a) Error Free Operation.



(b) Network Fault Delays All Messages.



(c) Losing Two Messages Allows All Others to Arrive on Time

Figure 5.1: TCAN: Simple Example Showing How Losing Messages Can Improve Timeliness of Others.

Section 5.5.

The basis of the protocol is that no part of a frame may appear on the bus after its delivery threshold/deadline $X_{i,k}$. This must be adhered to strictly, in order not to break further the atomic broadcast expectation.[3] This leads to the property that at (the global) time $X_{i,k}$, the frame $\langle i,k \rangle$ has either been transmitted successfully, or it will never be transmitted.

Since the worst case duration of each frame, C_i, is known, a *cut-off point*, termed the *latest send time* (LST), donated $L_{i,k}$ is defined. This marks the latest time that message transmission may *begin*, in order to ensure that if there are no faults then it is received by $X_{i,k}$:

$$L_{i,k} = X_{i,k} - C_i \tag{5.1}$$

In the following sections, the semantics of the transmitters and receivers are explained. Some simplified pseudo-Ada code is used (Figures 5.2 and 5.3) to show the semantics of TCAN. The code also shows how the protocol may be implemented in software, although a software implementation is not necessarily the best realisation. Section 5.8 discusses implementation strategies in more detail.

The example code uses Ada tasking, protected objects and the Ada select statement. It assumes a protected object (lst_interface) which the application and hardware use for communication with the driver task. For clarity, it only deals with messages of a single priority.

Transmitter Specification

For the most part, a TCAN transmitter acts exactly the same as a CAN transmitter: messages are queued and transmitted as usual and an error detected during transmission causes the frame to be re-queued.

The major difference in interface is that application software must additionally specify a delivery deadline (threshold) for each frame, in addition to the data.

From equation (5.1) the responsibility of a transmitter is to guarantee that the start

[3] Refer to Section 2.5.7.

of frame bit occurs no later than at time $L_{i,k}$. In other words, a frame must not attempt to enter arbitration after time $L_{i,k}$. The start of the frame may of course occur at any time before this.

At time $L_{i,k}$, if the frame has not already started transmission, the transmitter must have completed the removal of the frame from the transmission queue. It may correctly complete the removal before $L_{i,k}$; however this causes a performance loss. An implementation should therefore aim to abort a frame as late as possible, but always before $L_{i,k}$.

The action of the transmitter may be implemented by the Ada code in Figure 5.2.

```
1  task body lst_sender is
2    m : frame;
3    begin
4      loop
5        lst_interface.readmsg(m);
6        candevice.send(m.data,m.id);
7          -- write message to device for transmission
8        select
9          lst_interface.sent; -- protected object guarded entry which only
10           -- opens when the hardware interrupt has occurred, meaning
11           -- that the message has been correctly sent
12         lst_interface.markaborted(m); -- set fi eld in the frame so the
13           -- application knows that it was sent
14       or
15         delay until m.L - delta_t;
16         candevice.doabort(m.id);
17           -- send abort to device (if the frame is currently being
18           -- transmitted then it is not removed however it is not
19           -- requeued if there is an error)
20         lst_interface.markaborted(m); -- set fi eld in the frame so the
21           -- application knows that it was not sent
22       end select;
23   end loop;
24 end lst_sender;
```

Figure 5.2: Code for TCAN Transmitter.

In order to ensure that a message never begins transmission after L, it is necessary to adjust the delay in line 15 by some value δ_t (delta_t) to take into account execution time and delays. The instruction to remove the message from the hardware must not be executed late, (however it may be removed earlier than $L_{i,k}$).

Therefore, the delay must be until only $L_{i,k} - \delta_t$ where

δ_t is the worst case time that it can take for the second branch of the select statement to execute including the worst latency of the delay until statement.

Receiver Specification

A TCAN receiver acts like a CAN receiver for all parts, except that the application may also specify the deadline by which it wishes to receive the message.

If such a deadline is specified, then a receiver must ensure that it accepts any frame that begins transmission at any time up to L. As the transmitter will never send a frame after L, there is no penalty in listening after L, except for an obligation to the receiving application to (timely) notify that there was no frame.

Receiver code appears in Figure 5.3.

```
1   task body lst_receiver is
2     m:frame;
3   begin
4     loop
5       lst_interface.getidtoreceive(m);
6       candevice.listen(m.id); -- configure CAN hardware
7       select
8         lst_interface.received; -- guarded protected object entry
9         -- which only opens when a message has been received correctly
10        candevice.read(m); -- read from device
11      or
12        delay until m.LST + Ci + delta_r;
13        candevice.stop_listen(m.id);
14      end select;
15    end loop;
16  end lst_receiver;
```

Figure 5.3: Code for TCAN Receiver.

In order to guarantee that a receiver will accept any message up to the deadline, the delay must be increased in line 12 by a value δ_r (delta_r): where

δ_r is the worst case time that it can take to execute to the end of the first branch of the select statement, starting from the time of the end of the last bit of a frame.

5.4 Properties

The TCAN protocol described in Section 5.3 is extremely simple. Yet as Section 5.9 will show, the effect of it is to provide significant improvements to the number of messages that arrive on time in a flexible communications system. In general, using TCAN delivers approximately twice as many frames on time as CAN does under high-fault conditions. This section provides an insight into why it works, then explores the properties that any system based on TCAN can expect from the communication subsystem.

5.4.1 Insight

TCAN works by exploiting the fact that in an event-triggered system, tasks (or message frames) tend to suffer different levels of interference on each iteration. The interference depends on the relative periods of the higher priority tasks (or frames). This was explored in Chapter 3. In particular, with reference to Figures 3.1 and 3.2 on pages 78 and 79, note that there is only a relatively small number of peaks.

In CAN, instances of a frame suffer more interference (from higher priority frames) near to a critical instant. Where faults affect frames near the critical instant, frames may be pushed beyond some deadline. However, away from the critical instant, the same pattern of faults may not be sufficient to delay the frames beyond the deadline. Therefore, although the system may be able to function safely with some faults for most of the time, when the faults occur near the critical instant there is insufficient time to recover from them. It is these frames that the TCAN protocol does not transmit, thereby transmitting frames only if there is sufficient time to do so.

5.4.2 Properties of the Communication System

Never attempting to send a frame if it cannot arrive on time will provide a service with:

- **Real-time error confinement:** because there are no late messages, a late message cannot adversely affect the timing of any further messages, allowing the

bus to quickly "catch up". Compare this with normal CAN where after errors, a congested bus must first transmit all the late frames.

- **Timeliness:** timeliness is independent of faults. No matter how many network faults there are, messages will not be delayed beyond X_i—at the end of the window, the frame is lost.

- **Predictability:** Chapter 4 discussed why CAN cannot provide a predictable worst case delivery time for messages in the presence of faults. By eliminating all late messages, an element of predictability is provided on the bus. For each message, a well defined window exists, during which the message may appear. At run-time, TCAN ensures that the message cannot be on the bus outside of this window.

- **High reliability:** by maintaining the flexibility of CAN and allowing frames to be retransmitted at any time when the bus is available (except when a frame is late), the natural robustness of CAN to errors is not lost. In Section 5.9 this is demonstrated by means of simulation.

- **Atomic Broadcast is unaffected:** with the exception of the effect described in Section 2.5.7 which can prevent guaranteed atomic broadcast, either all receivers will receive the frame or no receivers will; the addition of TCAN does not change the existing semantics of CAN in this respect.

- **Common Knowledge of message delivery:** the delivery threshold deadline is known *a priori*. Therefore, at time X_i, all nodes know whether a frame has been received by all nodes, or whether the frame will never be received by any node. This is handled by the protocol, therefore the application never has to deal with late messages, only message omissions. Note that this property only applies under certain circumstances, described in Section 5.6.

- **Reliability of delivery not ensured:** Reliability of delivery is dependent on the nature of the errors and the amount of spare bandwidth reserved for errors. This requires the application designer to carefully consider the failure actions of a message not being delivered, but this is a good thing and is helped by the

fact that there is always a known point in time where the system knows that a message is definitely not going to arrive.

Note that while there are no faults and assuming that appropriate timing analysis has been performed, TCAN works exactly as for normal CAN: all messages are assured and timely. Only in the presence of faults may some messages be lost in order to ensure timeliness. As shown later, the chance of actually losing a message is small because in general only a small number of messages actually arrive close to their threshold times.

5.5 Strategies for Deadline Calculation

A significant number of policies emerge for which frames to abort and which to transmit. TCAN uses the concept of a delivery threshold or deadline for a frame. This section discusses several different schemes for setting the delivery thresholds and contrasts their effects.

A great deal of flexibility is available in deciding $X_{i,k}$. There may be a system-wide policy, different policies for each message stream, or even different policies for each individual frame. It should be noted that, ideally, thresholds are known to both the transmitter and the receiver. Therefore, the most useful thresholds may be ones that can be determined off-line, or inferred at run-time without further communication between nodes.

The threshold deadlines may be calculated dynamically, based on some suitable algorithm, or may be static in nature. Only static thresholds of the form:

$$X_{i,k} = S_{i,k} + X_i$$

where X_i is a constant for the stream are considered in the following because they are effective and easy to reason about. Dynamic deadlines (not of the form $X_{i,k} = S_{i,k} + X_i$) lead to extra complexity for little practical benefit.

5.5.1 Application Driven

To set frame thresholds, it may be useful to appeal to the applications' needs: a study of transactions or acceptable latencies within the application may prove a useful guide. However, setting $X_i = D_i$ is not necessarily the best course of action because

- in many industrial scenarios, there is often little understanding of why particular deadlines are used or how they are derived; this is little justification for using such values here, where the thresholds are directly related to the reliability of the system;
- better performance may be achieved with alternative values.

5.5.2 Manual Flexibility

In general, increasing the deadline for a frame increases the chance of delivery for the frame (since the size of the window in which it may be transmitted is increased). The cost of increasing X_i is the potential to delay lower priority frames.

Conversely, earlier deadlines (lower values of X_i) allow the bus to recover faster from faults and ensure that the particular frame has less effect on lower priority frames. This comes at the expense of an increased probability that the particular frame will be lost.

The deadlines may be manually adjusted to give more time for more important messages. In the case of a vital message, the threshold could be set to infinity so that the message will eventually get through if it can, even if it means losing other messages.

Consider a periodic sensor reading. If a process misses one reading then it may be appropriate for the receiving application to assume some value until a valid reading actually arrives, since control laws are generally able to cope easily if a data value is lost. A mode-change message however should not normally be lost since it is expressing a *change*. Therefore, regardless of the priority of the message, the importance of the mode change message may be expressed by increasing its delivery deadline.

Note that the CAN 'priority' (ID) is not the same as 'importance'. Messages can generally be assigned a 'priority' based on either *rate monotonic ordering, i.e.* how *often* the message should be transmitted; or by *deadline monotonic ordering i.e.* how *soon* the message should arrive as viewed from the application's perspective. The 'importance' of a message is related to how *necessary* it is for the message to arrive. In normal CAN, increasing the priority of a message does not increase the chance of it arriving (although it may increase the chance of it arriving before its deadline under fault conditions).

5.5.3 CAN $\subset$ TCAN

As the deadline of a frame is increased, the probability of the frame being successfully transmitted increases. The limit is where the deadline is set to infinity—which corresponds to no timeout. This is exactly equivalent to normal CAN.

Therefore, TCAN is a superset of CAN, meaning that any CAN system can be implemented using TCAN. Hence, TCAN is regarded as an extension to CAN (rather than a modification).

This may be considered as two general classes of messages:

A. $X_i < \infty$, non-assured delivery, but guaranteed worst case delivery time;

B. $X_i = \infty$, (almost[4]) assured delivery, but frames may arrive late (normal CAN).

The two classes may be safely used on the same bus since the addition of a timeout does not affect the functional operation of CAN. Note however that the use of class **B** messages at higher priorities than class **A** messages may lead to more class **A** messages being lost in the presence of high levels of faults.

5.5.4 Global Strategies for Deadlines

As there is a range of possibilities for deadlines, they may be arranged in a scale, ranging from 0 to infinite, see Figure 5.4. With the exception of $X_i = 0$, each range

[4]Reliable delivery with the usual restrictions of CAN, such as the Atomic Broadcast problem and bus partitioning. Both of these are regarded as outside the fault model here.

has a useful function.

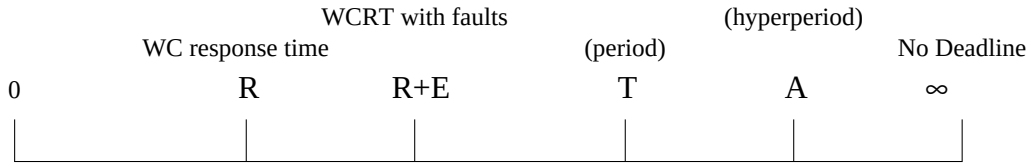


Figure 5.4: Possible Values for Deadlines.

In the following text, particular points and regions of interest are explained in a non-chronological order.

$X_i = \infty$ As discussed previously, an infinite delivery deadline (meaning no timeout) corresponds to normal CAN. This provides assured delivery, but with none of the advantages of using a timeout.

$X_i = T_i$ At the next level, the deadlines could be set at the period of the message (for a periodic message). Simulations reveal that this is actually an extremely effective approach overall, with only a very small fraction of frames being aborted, even for high levels of faults. However, this policy tends to have an extremely biased distribution of aborted priorities—nearly all aborted frames have the same priority. This is because having long deadlines does not provide much *error confinement*, delays can propagate down the priorities just like normal CAN. Eventually, the delays are absorbed by a frame where the response time happens to be close to the threshold.

$X_i = R_i$ Setting the deadline to be equal to the worst case response time is an extremely useful scheme, as will be demonstrated later.

Consider a bus with no faults, the worst case response time, R_i, can be calculated. It represents the longest time that a frame should take to reach its destination. If a frame takes any longer than R_i then it is clear that there have been some faults on the bus. Therefore, if the deadline is set equal to the worst case response time, then the TCAN protocol acts as a *run-time guard against faults causing delays on the bus*.

It might also be expected that using $X_i = R_i$ would be effective at removing bias in the distribution of aborted messages over the different priorities so that one particular application is not penalised, since each frame has a similar level of 'slack' time before the threshold.

On the other hand, a shorter deadline would increase the number of aborted messages. $X_i = R_i$ may be too small to be useful in the general case.

$X_i = R_i + E_i$ In a noisy environment, or if lost messages are more critical, time redundancy may additionally be used to tolerate a specified level of errors. In an effort to increase the deadlines from $X_i = R_i$ it is possible to set the deadlines to slightly higher than the worst case response time with no faults.

One way to do this is to incorporate redundancy into the worst case response time analysis using a bounded fault model, such as the sporadic fault model, equation (2.10) in Section 2.6.4 or (for example) Punnekkat's equation for $E(t_i)$ [131]. If the response time analysis shows that the system is schedulable then while errors are within the fault model used to calculate $E(t_i)$, reliability, predictability and timeliness are maintained. Only if faults at run-time exceed the fault model used in analysis is there a chance of losing a message (in which case timeliness and predictability are maintained).

In all cases, the TCAN extension acts as a *run-time guard to maintain timeliness and predictability if faults exceed the fault model used for analysis*.

$0 < X_i \leq R_i$ It is possible to set delivery deadlines that are less than the worst case response time. Of course if a deadline is less than the worst case response time then this does not guarantee that frames will arrive before their deadlines, and in the case of TCAN, this does not guarantee that frames will arrive at all. However, a threshold that is lower than the worst case response time can be useful.

Recall from Chapter 3 that if the frames have no-harmonic periods then huge variation is introduced into the response times of individual frames, with the worst case usually much worse than the average case. Therefore, a threshold less than the worst case (but greater than the average) would allow most messages to be sent, but would not send messages in the worst case.

This may be an effective means of increasing bus utilisation, particularly in non-critical systems. Additionally, it could be used to handle transient overloads in a predictable manner.

5.5.5 Weakly Hard and Probabilistic Guarantees

One of the most useful approaches for setting the delivery threshold policy is to combine TCAN with a combination of the weakly-hard and probabilistic analyses from the preceding chapters.

Weakly-hard Analysis

The multiple invocation analysis in Chapter 3 provides the worst case response times for all invocations in the hyperperiod. Using this analysis, and a specification of required performance (expressed as weakly-hard constraints) a value of X_i can be chosen such that the minimum weakly-hard specification is maintained.

As a trivial example, suppose that the set of worst case response times (incorporating some fault model) is $\{400, 200, 100\}$ and a weakly-hard specification of $\binom{2}{3}$ is required, then a threshold $X_i = 200$ will guarantee the constraint (if faults do not exceed the model used for analysis) but will not attempt to waste bandwidth on transmitting this frame near a critical instant where the response time may be up to 400. To increase the probability of the constraint being satisfied if faults do exceed the model used in analysis, then the value of X_i can be increased using probabilistic analysis.

Probabilistic Analysis

The probabilistic analysis in Chapter 4 can be used to explore what-if scenarios and derive an appropriate value for X_i to ensure a sufficient probability of delivery.

Using an estimation of the fault rate, the probabilistic analysis will generate a probability distribution for the response time at the critical instant. Assuming that a desired probability of meeting the deadline can be specified, an appropriate value for the delivery threshold can be read directly from the distribution.

By using this technique, a high level of fault confinement can be provided through the use of TCAN, yet the system is analysable and shown to be capable of supporting the minimum requirements for delivery.

5.6 Common Knowledge of Delivery Threshold Times

So far it has been assumed that a global delivery deadline $X_{i,k}$ is known to the transmitter and all receivers of a particular frame. In some cases this is possible, in others it is not. This section first discusses the circumstances where it is possible to determine a global deadline for a frame, then the use of TCAN when it is not possible to set a global deadline is explained.

As an example of a class of message where a global threshold value is known, a *periodic* message stream is considered. As an example of a class of messages where a global threshold cannot be determined, *sporadic* messages are used.

5.6.1 Periodic Messages

For purely periodic messages in a stream, the following are assumed to known *a priori*:

- the period, T, of the message;
- the offset O_i of the stream: the time that the first message in the stream is released relative to a common event such as an epoch.

Based on the common knowledge above, the threshold for a periodic message, $X_{i,k}$, can be calculated:

$$X_{i,k} = O_i + kT_i + X_i \tag{5.2}$$

$X_{i,k}$ and hence $L_{i,k}$ are therefore known to all nodes in the system. The advantage of common knowledge of $X_{i,k}$ is that at this time, all receivers must have either received the message or know that the message will never arrive. Therefore, any receiving application may begin error recovery actions for a lost/late message at a known point in time.

The above parameters are in accordance with the widely used periodic model. However, although time-triggered real-time systems necessarily require an offset, event-triggered real-time systems are often designed without using any explicit form of offset, even though an offset may implicitly exist. If it is not possible to determine the offset then it is not possible to determine a global deadline without some further communication between nodes; instead treating the frame as non-periodic (see section 5.6.2) may be useful.

5.6.2 Non-periodic Messages

A non-periodic message may be queued at any time, only known to the transmitter. So in a distributed system with TCAN there is no timely way for a receiver to know that the message is queued for transmission, hence no way to know the deadline. Therefore receivers cannot know when to stop listening for a given message. Two ways of using non-periodic messages with TCAN are explained below.

No timeout: One way is to consider the nature of non-periodic activity. Non-periodic messages tend to occur infrequently and tend to contain data which indicates an event or a state-*change*. For these messages, simply setting the delivery threshold to infinity (normal CAN semantics) may be more appropriate so that a non-periodic message will never be lost due to timeout.

Using this method, schemes often used in normal CAN such as time-stamping the data may be useful to allow application to know when the message was sent.

Asymmetric Scheme: Alternatively, for non-periodic messages where the data is not critical or where the data will expire, it is better to use a timeout in order to provide error confinement. An asymmetric scheme may be used where:

- the transmitter does enforce a delivery threshold;
- but the receivers *do not* have a timeout.

This asymmetric treatment of non-periodic messages means that messages will not arrive ‘late’ with the benefits of error confinement and preventing further delays on the bus.

However, there is an important semantic difference compared to periodic messages. For a late non-periodic frame, the receiver application has no idea that a sporadic transmission attempt was made, whereas for a periodic message all receivers infer that there was a periodic transmission that failed due to faults.

5.7 Clock Synchronisation

Previously, in order that each node can know *a priori* the delivery threshold time of each message frame, there has been the assumption that all clocks are exactly synchronised. Clearly, this is unrealistic. In this section, the requirement for exact clock synchronisation is relaxed. Two schemes are explained, first a method by which approximate clock synchronisation is sufficient; second, a scheme is provided such that if the clock synchronisation assumptions are broken at run-time, then the TCAN protocol can still function correctly.

5.7.1 Effects of Clock Skew

It is the responsibility of the sender never to begin to send a message after the latest send time, $L_{i,k}$, has expired. However if a receiver's clock is faster than the sender's clock then the transmitter may send the message too late for the receiver to receive it. Therefore the receiver will reject the message while others with slower clocks may accept it, as shown in Figure 5.5. This may or may not be a problem for a particular system, but it provides opportunities to further break atomic broadcast in CAN.

Section 2.5.8 introduced means of providing clock synchronisation on CAN. The schemes were divided into two classes: software implementations with a low accuracy of synchronisation, and schemes which use hardware support to provide a higher degree of accuracy. The symbol ε is used to mean the bounded clock difference: *i.e.* the maximum simultaneous difference between any two clocks in the system.

The two general ways of dealing with clock skew are called *unprotected-TCAN* which is mostly suitable for small ε (low clock drift), usually when hardware clock synchronisation is used. It requires the clock drift to be *guaranteed* to be less than ε.

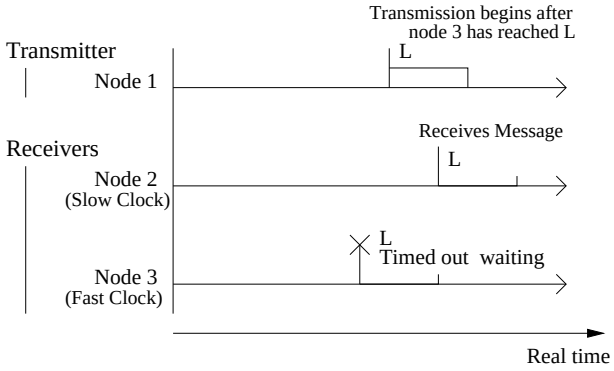


Figure 5.5: Effect of Clock Skew.

That is, if the clocks happen to drift beyond ε then inconsistent scenarios may be observed where frames are accepted at some nodes and rejected at others.

The other mechanism, called *protected-TCAN*, is better suited to larger values of ε, where such accurate synchronisation is not possible. Its correctness is not dependent on any value of ε—if at run-time the clocks happen to drift beyond ε then this will not lead to inconsistent scenarios—although very large run-time clock drift will result in performance loss.

5.7.2 Unprotected-TCAN

The basis of the unprotected-TCAN scheme is to ensure that even if a receiver's clock is late, and the transmitter's clock is early, then receivers never miss a message. To guarantee this, each *receiver* makes an adjustment for bounded clock skew: all receivers accept messages until the local time $L_{i,k}^{local} + \varepsilon$. Transmitters still continue to transmit until $L_{i,k}^{local}$. Therefore there is a window during which any difference in clock synchronisation will be compensated.

This can impose more traffic on the bus than was intended, however the extra traffic is extremely small and will not significantly affect performance. In any case, clock synchronisation may be included in worst case response time analysis to guarantee schedulability. This is a practical way of dealing with clock skew. This mechanism is called unprotected-TCAN.

In theory, it does not matter whether transmitters or receiving nodes adjust the window; they are exactly equivalent, the difference being that the overall value of $X_{i,k}$ is assigned a different value. Although it is possible for the *senders* to make the adjustment: senders do not attempt to transmit a frame after time $L_{i,k}^{local} - \varepsilon$, so a receiver should listen for the start of the message until time $L_{i,k}^{local}$, assuming the worst case skew like this could lead to messages with short response times never being transmitted even under normal conditions. It is preferable for receivers to make the adjustment by extending their window by ε.

5.7.3 Protected-TCAN

The Protected-TCAN mechanism involves a further change to the protocol but provides a more robust way of dealing with clock skew. This mechanism is particularly suited to larger values of ε or where ε cannot be guaranteed.

If a receiver receives the start of a message after $L_{i,k}$ for that message has passed (according to its local clock) then it is an error. Errors are signalled to all other nodes on the bus by transmitting a CAN error frame immediately after the complete arbitration field has been sent. Therefore all other nodes will reject the frame and the sender will abort transmission (see Figure 5.6). This mechanism is called *Protected-TCAN*.

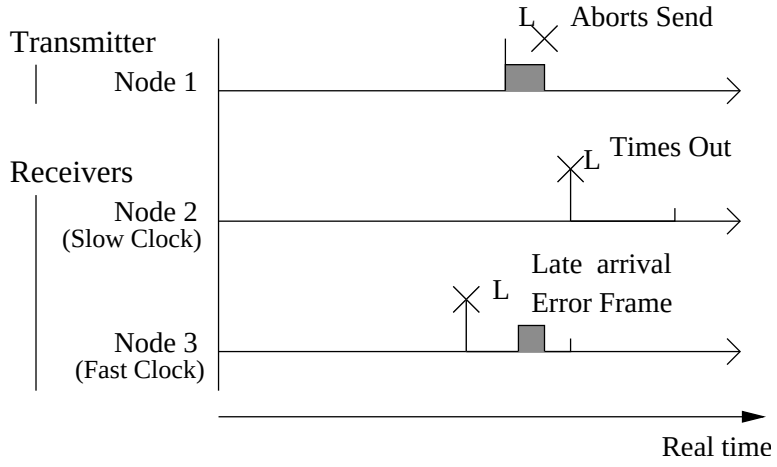


Figure 5.6: Protected-TCAN: A Late Message Generates an Error Frame So No Nodes Receive the Message.

The advantage of Protected-TCAN is that the protocol still preserves the same level of atomic broadcast as CAN, even if the clock synchronisation fails to stay within the assumed precision. This is important since clock synchronisation relies on communication between the nodes, which is not guaranteed in the presence of faults. Some level of synchronisation is still required, as very large differences in clock synchronisation will result in performance loss because more messages will be aborted, but if the bus is so noisy that not even the clock synchronisation messages get through, then messages lost due to clock drift are insignificant compared to the messages lost as a direct consequence of the interference itself.

5.8 Implementation

Implementation of the protocol does require modification of the CAN protocol, but changes are minimal and can be seen as extensions, rather than deviations.

There are two suggested implementation routes: either a custom hardware controller, or for some COTS hardware, CAN driver software may be modified to provide TCAN.

An approach such as FTTCAN [3] could also be used to implement timeouts in CAN. However, TCAN is intended as a small change to CAN, and it is better implemented without the network overhead or complexity of FTTCAN.

In practice, custom implementation of a TCAN controller may be the best implementation and in critical systems this may well be done as a matter of course. Nowadays, the availability of dedicated single chip computers and FPGA hardware makes implementing a custom controller feasible for smaller projects.

Nevertheless, the protocol can be implemented in driver software using current COTS CAN controller hardware such as the Philips SJA1000 [124] or Intel 8257 [71]. Note that not all CAN controllers are capable of supporting TCAN, for example the Motorola TouCAN [109] controller is not capable because of its internal buffer arrangement. The CPU overhead of a software implementation is fairly low and merely involves removing messages from the queue if the deadlines expire.

A TCAN protocol implementation may be based on either BasicCAN or FullCAN interfaces. The difference between BasicCAN and FullCAN is that in BasicCAN the hardware is much simpler, typically providing only one transmit buffer and minimal receive filtering. A FullCAN controller does much more in hardware, usually providing message "objects" which can be individually programmed to transmit or receive particular frames.

The most difficult aspect of the implementation is dealing with clock skew: if accurate clock synchronisation is not available, then it is appropriate to use Protected-TCAN (see Section 5.7.3). This requires modification to the CAN hardware. Immediately upon receipt of certain message IDs, the hardware must transmit an error flag to the bus. Although this in itself is not a difficult task in a hardware implementation, the controller must have exact ID matching/filtering which few controllers currently provide.

If accurate clock synchronisation is ensured then the implementation of the receive operation is considerably easier because unprotected-TCAN may be used. This can be done entirely in software. The TTCAN synchronisation mechanism is suggested as an appropriate hardware supported clock synchronisation method. This is not difficult to implement in custom hardware. Alternatively, future COTS CAN controllers are likely to support this directly.

It should be noted that the TCAN protocol is ineffective if the controller suffers from problems such as inner priority inversion (where a node continues to attempt to transmit a lower priority frame in its buffer even when a higher priority frame becomes queued for transmission on the same node). Mechanisms for ensuring correct priority scheduling are often supplied in application notes for individual COTS controllers [124].

A hybrid implementation using an FPGA and a COTS CAN controller is underway [57]. This uses the CAN controller to manage normal message transmission, and programmable hardware to keep track of delivery timeouts and clock synchronisation efficiently.

5.9 Evaluation

This section evaluates the effectiveness of TCAN through fault injection modelling using the software CAN simulator previously described. The simulation aims are:

- to test the ability of the protocol to confine faults;
- to measure the reliability of delivery provided by the protocol.

5.9.1 Model for Simulations

As in the previous chapters, the simulator is set to model worst case message length and bit stuffing, and accurate modelling of faults at the bit level.

The (harmonic) SAE message set (see page 91) is used again as a benchmark in the following tests. Recall that for some messages even one fault is sufficient to cause a deadline failure.

In the simulations, single bit errors and burst errors are injected onto the bus. Single bit errors affect exactly one bit on the bus; burst errors affect the bus for 1000 μs (125 bits). The faults were distributed randomly according to a Poisson distribution.

Each simulation was repeated for normal CAN and for TCAN protocols. The faults were injected at exactly the same times for the CAN and TCAN simulations. Each simulation was performed for 100 times the hyperperiod of the messages. The first frame of each stream was released at time $t = 0$ plus some random jitter in the range specified in Table 3.1.

5.9.2 Testing Fault Confinement

The first set of simulations explores the overall ability of TCAN compared to CAN to deliver messages on time during very high levels of faults that exceed normal working conditions. *Success of delivery* is measured using a simple count of missed deadlines and lost messages.

Simulations were performed at a variety of fault rates, incorporating single-bit and burst errors. For the TCAN experiments, the delivery threshold is set to the deadline, $X_i = D_i$ (which is equal to the period, T_i, for most of the streams). This assignment of X_i makes an explicit equivalence between late and aborted messages; other assignments are possible of course where the delivery threshold does not equal the deadline.

The results are summarised in Table 5.1 for 13 different fault models. The 'Late Frames' column shows the number of frames that arrived late due to faults in normal CAN *per second*. The 'Aborted' column shows the number of frames that were not delivered according to the TCAN protocol for exactly the same fault scenarios. The percentage columns contain the late/aborted frames as a fraction of the total number of messages sent per second. The rightmost column $\frac{Aborted}{Late}$ shows the ratio of messages that were lost in TCAN compared to the ones that arrived late in CAN.

Table 5.1: TCAN and CAN Fault Injection Results. All Values are for 1 Second.

Key	**Bit errs.**	**Burst errs.**	**CAN Late Frames**	**%**	**TCAN Aborted Frames**	**%**	$\frac{Aborted}{Late}$
A	20	0	0.13	0.009%	0.13	0.009%	1.000
B	40	0	0.78	0.054%	0.73	0.051%	0.936
C	80	0	3.88	0.269%	3.53	0.244%	0.910
D	120	0	11.55	0.800%	9.29	0.643%	0.804
E	160	0	24.45	1.693%	18.79	1.30%	0.769
F	200	0	45.21	3.13%	31.92	2.21%	0.706
G	260	0	100.82	6.98%	55.57	3.85%	0.551
H	320	0	16.406	11.4%	84.17	5.83%	0.513
I	10	5	0.11	0.008%	0.11	0.008%	1.000
J	20	10	0.43	0.030%	0.42	0.030%	0.977
K	40	15	2.07	0.143%	1.93	0.133%	0.932
L	80	20	8.36	0.579%	7.19	0.500%	0.860

Frames queued per second = 1444

For occasional, single bit errors, the simulations show that the TCAN and CAN protocols perform very similarly. In fact, for model 'A' (20 single bit errors per second), in both simulations exactly the same messages missed their deadlines (or were aborted in the case of TCAN). It is interesting to note that even though the analysis cannot guarantee all frames will be timely, the actual proportion of frames which do

not arrive on time is extremely low (0.009%).

For high error rates the simulations indicate that the TCAN protocol provided significant improvement over CAN in terms of number of late (lost) messages. For fault model G for example, approximately one third as many frames were lost in TCAN than arrived late in CAN, showing the capability of the protocol to maintain a high reliability during exceptional levels of faults.

It is clear that where the faults are very high, in order to prevent the bus from lagging further and further behind, it is necessary to abort some retransmissions, which TCAN does. Of note from this experiment:

- it is perhaps surprising that the number of aborted/late frames is so low: only 0.25% of frames were aborted at the fairly high error rate of 80 single bit errors per second;
- most of the frames that were lost were of the same priorities (12 and 8). This very skewed distribution is due to the fact that the worst case response times of those messages is close to their delivery deadlines, hence only a few faults are required before the threshold is exceeded. For model E, 1239 out of the 1879 aborted messages (66%) were frame 8. The same is true for CAN, 1643 late out of the 2445 (67%) were frame 8.

The skew noted in the experiment is highlighted in Figure 5.7 which shows the distribution across priorities of aborted frames.

5.9.3 On Removing Bias

Faults anytime in the system can cause a domino effect, delaying all frames until one of them exceeds the deadline, at which time the bus is allowed to catch up. Where thresholds are close to the worst case response times, those particular frames are the ones that are lost. The TCAN protocol is able to reduce the skewed distribution of lost messages by considering a less naive threshold strategy than simply using $X_i = D_i$.

Section 5.5 provided a number of policies for determining threshold values. In particular, it noted that setting the threshold to some function of the worst case response

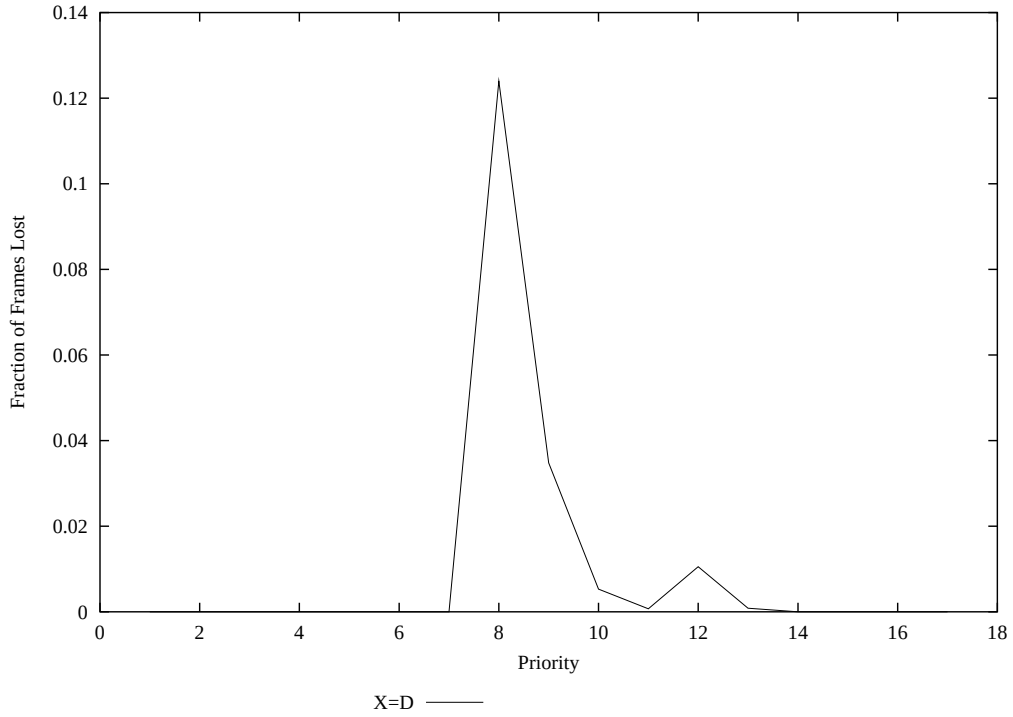


Figure 5.7: Distribution of Lost Frames for Model F.

time is useful. Repeating the experiments with $X_i = R_i$ (where R_i is the worst case response time not considering any faults) gives stream 8 some reprieve from missing deadlines, but at the cost of losing some other frames.

However, the very low priority frames lose considerably more frames than before due to their shortened thresholds. Note that the period of the highest priority message is larger than most other frames, therefore frame 17 may only interfere with the lower frames occasionally (once per second); whereas for the lower priority frames, the frame 17 always interferes. It is worth noting that including the highest priority message in the analysis is almost the same as including an error term. If the highest priority message is removed from the simulation (and the response times lowered accordingly) then more frames are lost/late than if the frame is kept in.

Ideally, to increase the chance of an individual frame being received, a larger value of X_i should be used to include some overhead for faults as previously discussed. However, as this message set cannot guarantee to tolerate even one fault for frame 8, then it is not possible (or useful) to increase the threshold of the higher priority frames.

Nevertheless, it is possible to use a hybrid scheme: for priorities 17–8, let $X_i = R_i$, for the others, let $X_i = D_i$. This results in a slight flattening of the distribution, see Figure 5.8, but as there is little slack in the deadline for frame 8, there is very little that can be done.

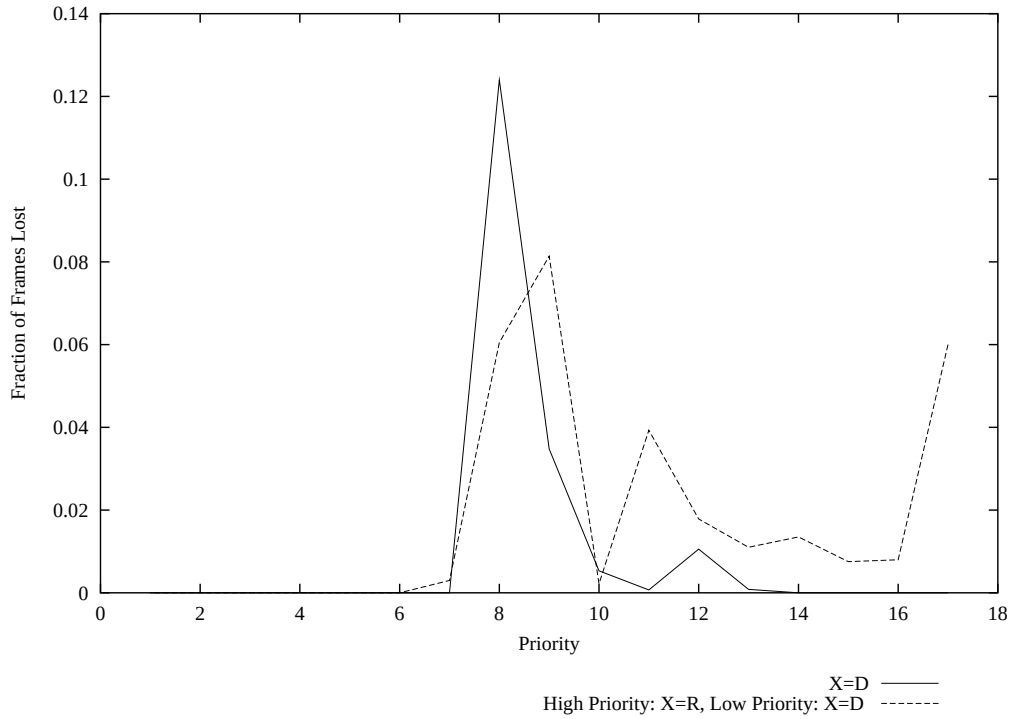


Figure 5.8: Distribution of Lost Frames for Model F with 3 Threshold Schemes.

5.10 Summary

The Timely-CAN (TCAN) protocol presented here is a small extension of CAN which enforces predictable timing behaviour on CAN in the presence of unspecified transient network faults. It prevents network faults affecting the timeliness of messages by sacrificing guaranteed delivery. It can be used to help systematically provide operational status with degraded performance in the presence of unknown network errors. The protocol is simple to understand and to implement, and the run-time CPU overhead is small.

TCAN provides real-time predictability by guaranteeing that *late* messages are not allowed on the bus. Each frame has a delivery deadline (or threshold); various schemes were discussed for setting the deadlines. One suitable scheme was highlighted: setting the deadline to be equal to the worst case response time including some redundancy for faults. This ensures that while network-born faults are within the fault model assumed for worst case response time analysis, the protocol behaves exactly as normal CAN. However if errors exceed the fault model (possibly introducing unanticipated delays) then any late messages are not sent on the bus. This can be seen as a form of real-time error confinement, allowing the bus to recover quickly from faults.

Messages queued for transmission may not necessarily arrive, therefore applications must be able to cope with lost messages. However, the sender and receivers always know whether or not a periodic message has successfully been sent. At a globally known point in time, all nodes will either have received the frame, or they will know that the frame will never arrive. This is an advantage for system design because it is generally easier to reason about lost messages than it is to consider messages which may arrive at some unknown time later. For example, assumptions about bounded size buffers still hold [46].

Normal CAN messages (which do not have a timeout) can also be transmitted simultaneously on the same bus. They are *assured delivery* messages (to the same extent that CAN normally provides assured delivery) but the timing behaviour of such messages in the presence of unpredictable faults is unpredictable—the same as for normal CAN. However, some speed-up might be expected if there are higher priority messages which do have a delivery threshold.

Simulation shows that the protocol is effective at providing error confinement and at improving the overall reliability of the bus (measured in terms of the number of frames delivered on time). Results with the SAE benchmark indicate that the overall success of delivery of TCAN is high even for exceptionally disturbed buses. An additional example showed how the protocol can work in conjunction with weakly-hard and probabilistic analysis.

The TCAN protocol has been shown by these examples to give a considerable increase in quality of service on CAN under high levels of faults. Furthermore TCAN

still allows the flexible nature of CAN to be used to accommodate errors, leading to an extremely reliable (nearly all messages get through) yet predictable bus in the presence of network faults.

TCAN can be considered to lie somewhere between CAN and TTCAN: it exploits the natural robustness of CAN by allowing message retransmission. However it only allows the (re-)transmissions within a specified window, the width of which is determined by response time analysis. This provides the timing predictability of TTCAN and yet ensures that the CAN re-transmissions are used to good effect to provide a very reliable bus.

6 Conclusion

THE ABILITY TO MAKE SCHEDULING DECISIONS AT RUN-TIME is a key aspect of flexibility that may be exploited to provide dependable real-time communication. Such flexibility has the potential to allow a bus to achieve efficient fault tolerance and support non-periodic data or messages with partially known timing requirements, attributes which are of increasing concern in distributed control systems.

The 'uncertainty' often associated with flexibility has been prohibitive in the use of a flexible bus in dependable systems: the *predictability* of a bus is important [146]. However even though it is not possible to predict the exact sequence of events on a flexible bus, it is not necessary to do so: the properties of the bus are predictable at a higher level—at the interface to the application, where a well-defined, *predictable* service can be provided.

Throughout, this thesis has referred to the unpredictable nature of faults and the difficulty of estimating faults which may affect a communications bus in the future. Flexible communication, making scheduling decisions at run-time, can be a powerful way of dealing with such uncertainty. Simulations and analysis have shown that high *reliability* can be obtained through the use of this fault tolerance.

This thesis has shown that flexibility may be exploited in event-triggered communication to provide a reliable, fault tolerant service which is capable of supporting dependable systems. This thesis has demonstrated this in the preceding chapters using a number of techniques including analysis schemes and run-time support. In the following, the themes in this thesis are summarised briefly and evaluated.

6.1 Missing Deadlines

The assumption that all contributions in this thesis share is that missing deadlines in real-time communication is not always catastrophic. Applications tend to exhibit robustness to a small number of timing faults: for many applications, occasional single missed deadlines can be safely handled and recovered from. The firm deadline model (as currently used in many avionics applications) is adopted.

Additionally, this thesis asserts that missing deadlines in real-time communication is:

Unavoidable since the faults that will affect the communications bus are hard (or impossible) to predict, *no electrical bus system* can guarantee reliable and timely delivery unless the future fault conditions are known.

Analysable but not constraining: the concept of a hard deadline provides a clean way of describing correct behaviour, however other forms of deadline (such as weakly-hard deadlines and probabilistic deadlines) are also well-defined ways of describing timing behaviour and with appropriate analysis may be used to predict behaviour.

Useful since although deadlines may be missed in the worst case(s), in most cases deadlines are still met, and missing occasional deadlines can be used to enhance the schedulability, fault tolerance and utilisation of a bus.

Missing deadlines therefore is more acceptable than is often implied by some other research in real-time systems. This thesis makes the step from hard deadlines (with their advantages in reasoning about a system) to other forms of deadline that are not hard, but yet are still analysable and provide practical advantages over systems designed using only hard deadlines. In particular, the firm deadline model is assumed with additional information about the number of deadlines that may be missed, such as weakly-hard constraints and probabilistic guarantees.

6.2 Exploiting Variation

In an event-based bus, it follows that the response times of the invocations of a message in a stream will vary according to the current traffic on the bus. Specifically, for CAN, this means the interference of higher priority frames varies for different invocations of the same message.

Although predicting the exact response times is infeasible, if sufficient is known about the *pattern* of interference from higher priority frames, then multiple invocation analysis can be used to derive a *worst case* response time for individual invocations. This then leads to weakly-hard deadline guarantees where specific bounds can be placed on the number and pattern of deadlines missed, as described in Chapter 3.

Where periodic messages with non-harmonic periods are used, then the frequency of near-worst case occurrences is much reduced, and the variation in response times is maximised. Although in a traditional real-time system, variation is considered difficult to deal with, weakly-hard analysis on CAN allows real-time behaviour to be analysed and predicted.

The advantage of using the analysis is that the overhead of a high number of faults can be incorporated into the calculations—much higher numbers of faults than normal worst case response time analysis can consider. This means that even though a bounded fault model is used in the analysis, the parameters of the fault model can be increased to consider the behaviour at higher fault levels. Therefore the probability of faults at run-time exceeding the fault model used for analysis is smaller. Hence more confidence can justifiably be placed in the behaviour and performance of the system.

6.3 Exploiting Dynamic Fault-tolerance

The advantages and the problems of the bounded fault model have been referred to many times throughout this thesis. A more realistic way to model faults is to consider a random fault model where faults occur at random times, following a known distribution such as the Poisson distribution. This leads to the concept that worst case response time analysis (in the usual sense) is not possible, because a known worst case

(*i.e. bounded* network activity) is a necessary condition for worst case analysis.

However, it is possible to exploit the fact that in an event-based bus, bandwidth may be used flexibly to retransmit corrupt data. Therefore the probability of a message being lost (or late) due to faults is not dependent on merely the probability of a frame being corrupted (as in the case of TDMA communication) but it is also a function of the amount of network activity near the time of the fault.

The effect of the dynamic fault-tolerance is analysable with a random fault model to provide the probabilistic worst case response time analysis introduced in Chapter 4.

One advantage of a probabilistic worst case response time analysis is that rather than an over-simple 'Yes/No' schedulability test, the analysis gives a quantitative indication of reliability to the designers, showing potential areas of failure where alternative fault tolerance may need to be applied.

6.4 Exploiting Missed Deadlines

For an application with explicitly firm deadlines, it is possible to further exploit the flexibility of a bus to make scheduling decisions at run time, by legitimately not transmitting messages that would arrive late. This frees bandwidth and allows the bus to recover from faults faster than if it had to first transmit queued messages. This run-time mechanism can be viewed as a real-time fault-confinement scheme, to protect against exceptional levels of faults that may occur.

The net result is that under even very high levels of faults, only a small number of frames need to be sacrificed in order to deliver other frames on time, providing a high reliability communications bus. Additionally, it allows the specification of an acceptable window of transmission for each frame. This means that the effects of any number of faults on the timing behaviour of the bus is limited to within the delivery deadline of queued frames only. An extension to CAN, named Timely-CAN (TCAN) has been suggested in this thesis which incorporates the notion of delivery deadlines into CAN.

6.5 Evaluation

This thesis forms only part of continuing research around the world into flexible real-time communication and its use in dependable systems. This research includes significant recent work on the reliability and timing properties of CAN [140, 117, 3, 55, 113, 57]. Collectively, this research provides a strong argument that a flexible bus may be used to provide communication in a dependable system. In particular, this thesis shows how the flexibility can be *exploited* to implement fault tolerance in a predictable way.

Where this thesis differs from similar research [126, 131, 142] on real-time communication is the acceptance that real-time computing is not the same as simply guaranteeing hard deadlines. Indeed, the starting point of this work is that it is not necessarily possible to *guarantee* deadlines at all in embedded communication because the external effects of environmental interference may be unpredictable [35]. Such an observation is frequently missed in analytical real-time research [127, 2, 96].

Therefore, the approach taken here, to accept deadline misses and to try to exploit the variation and flexibility of event-triggered communication, intrinsically leads to very practical results that may be particularly suited to industrial applications. Current best practice is to make timing behaviour 'guarantees' for CAN based on worst case response time analysis [159]. This necessarily involves accurately deriving a worst case error overhead function, $E_i(t)$, for each application through measurement or estimation. However, the difficulties of doing so (with any accuracy) are clear; unsurprisingly, few guidelines are available for such a derivation. In comparison with current practice, this thesis shows that:

- worst case response time analysis is not necessarily the best way to view the system, instead looking at the other (non-worst case) invocations and probabilities of failure can show a (perhaps surprising) level of reliability;
- a bounded $E_i(t)$ is not necessary, instead only approximate estimations of faults are required to gain an understanding of the probability of failure.

The acceptance of a firm deadline model (as normally used in ARINC-629 communication in avionics [8]) rather than considering only hard deadlines, is one key to

releasing the flexibility of CAN. As Chapters 3, 4 and 5 showed, allowing a flexible bus to miss occasional deadlines under fault conditions can provide considerable advantages in terms of the ability to support practical levels of analysable traffic, and the overall reliability of communication.

It is easy to overlook the fact that dependability comes from the combination of reliability, timeliness, availability, safety and other facets. System dependability must be considered as a whole, taking into account the services and interactions of all parts. No bus can guarantee both timely and reliable delivery in the presence of unpredictable faults, so it would seem that some compromise between timely delivery and reliable delivery is required. The results of this compromise may be measured using metrics such as *reliability* and *availability*, with subjective evaluations of *predictability* and *safety*, leading eventually to a justifiable argument for dependability.

In time-triggered communication, including ARINC-629, the focus is mainly on timely communication; fault tolerance concerns are secondary. TTP for example implements tolerance against data corruption by routinely transmitting all data twice [83] to increase the probability of delivery. CAN on the other hand has increased potential to recover from faults efficiently (but at the cost of guaranteed timeliness). The *natural robustness* of a flexible bus may be used to good effect, message retransmissions need only take place where necessary, and as many times as necessary. Therefore, the use of a flexible bus, such as CAN, may improve system dependability through its efficient error recovery mechanism. To ensure other parameters of dependability, such as predictability, a combination of analysis and the TCAN protocol can be used.

The importance of being able to support partially known, or flexible, timing requirements in dependable communication is reflected in the design of FlexRay [19], the automotive industry's new protocol which is likely to succeed CAN for in-vehicle requirements. FlexRay, although largely TDMA based contains some slots reserved for event-triggered communication. The advantages and disadvantages of this hybrid approach have been discussed before in a variety of areas [100, 128, 14]. One particular motivation for using a TDMA foundation is to guard against any timing failures (whether caused by disturbances on the bus, or by faulty nodes). In many respects TCAN is able to provide a similar level of protection as a TDMA bus by enforcing acceptable windows for transmission, whilst maintaining the flexibility to support

partially known timing requirements. However, a CAN based protocol such as TCAN does not implicitly have the ability to detect faulty nodes which incorrectly transmit data to the bus (and hence potentially starve other nodes of sufficient bandwidth to meet their timing requirements. Recent work on the *babbling idiot failure* may help to ensure predictable timing behaviour even when nodes are faulty [34].

In summary, the approaches described in this thesis, supplemented by relevant examples and simulations, which in conjunction with other evidence mentioned throughout the thesis, provide strong argument for the use of flexible communication in dependable systems.

6.5.1 FlexRay

At the time of writing, FlexRay is beginning to take a stable form. Recall that FlexRay implements a TDMA schedule with slots reserved for event-triggered communication; within the event-triggered slots, a protocol called ByteFlight [123] is used to provide arbitration. The draft specification [19] seems to be generally accepted. The recent FlexRay workshop in June 2003 promoted the protocol as one which has been carefully considered to cater for the many needs of the members of the FlexRay consortium,[1] including support for deterministic behaviour, robustness against transient faults, application-controlled safety ("application is responsible for decision concerning vehicle safety/availability") and data consistency/agreement within distributed applications. It is relevant to briefly compare the results of this thesis with FlexRay.

The similarities between FlexRay and TCAN are notable. Since FlexRay is TDMA based, there is no inbuilt mechanism for selective retransmissions of corrupt frames. Therefore both FlexRay and TCAN (but not CAN) present the same 'firm deadline' interface to the application, a frame either arrives on time, or it does not arrive at all. This model supports the deterministic behaviour requirement, since for each message, a defined window in time can be stated during which the message must arrive.

TCAN and FlexRay both move fault tolerance upwards to the application level, making the application aware of the failure, although the extent to which this is done

[1] See the presentations online from www.flexray.com .

differs. The FlexRay documents state that an application should be able to tolerate several cycles with no message reception [19]. Since FlexRay does not make any attempt at retransmission of corrupt frames, it might be expected that the proportion of lost frames in FlexRay is higher than in TCAN which directly supports robustness against transient faults.

One fundamental difference between FlexRay and CAN is that the higher bit-rate of FlexRay means that the signal propagation time is greater than one bit time. The implication is that the protocol cannot guarantee that messages are delivered to all devices consistently [123]. Notably, the transmitter has no (inbuilt) way of discovering whether or not a frame was correctly received. Higher level agreement protocols are necessary if an acknowledgement is required [19]. In fact, neither FlexRay nor ByteFlight provide any support for retransmission of data (although a higher level protocol may be used to implement such a mechanism). This, in contrast to CAN and TCAN, reduces the potential flexibility of FlexRay to be able to tolerate corrupt frames through retransmission in order to provide agreement. Nevertheless, higher level protocols may well emerge which exploit the higher bit-rate of FlexRay (up to 10Mbit/s per channel) and use reserved event-triggered slots for selective message retransmission. There are numerous possibilities as this new technology evolves.

The analyses in Chapters 3 and 4 are specifically for CAN and are related to retransmissions in CAN, but it may be possible to adapt them for FlexRay with a higher level protocol, perhaps using techniques similar to those of Audsley in timing analysis of APEX (ARINC-653) [13].

6.6 Exploitation of Results

The three main contributions in this thesis are directly concerned with the CAN protocol and as such they are of interest to the wide audience of industrial users of CAN.

The two new forms of analysis for CAN presented in Chapters 3 and 4 may be applied directly to new and existing systems. These contributions are of particular interest to designers using CAN because their use does not require any changes to the design nor to the design process. Tool support could easily be provided in a way

that is compatible with existing tools since both forms of analysis are able to be fully automated (they do not require any interaction with the analysis process). An existing tool vendor may be the most appropriate route for exploitation.

Although the advantages of timeouts in TCAN may be viewed very favourably by the users of CAN, the TCAN protocol is much less likely to see widespread use than the analysis techniques. The reason is that the TCAN protocol usually requires a small change to the CAN hardware in order to be efficiently implemented, see Section 5.8. The existing level of industrial investment in CAN is very high, including an efficient high volume/low cost manufacturing process. To use any CAN-based protocol requiring hardware modifications would be costly, and a manufacturer would be unlikely to risk the investment for a non-standard chip. For small scale use, TCAN implementation in programmable hardware is a useful implementation route [138].

An alternative exploitation route would be through the standardisation process; the TTCAN protocol is currently being incorporated into the latest ISO standard for CAN [72] and will eventually become 'Part 4' of the standard. However, the standardisation progress is too far advanced for any additional changes. Hence TCAN is unlikely to be recognised as a standard. In addition to the above, now 15 years after its introduction, CAN may be regarded as an old protocol, to be replaced (in new automotive applications at least) by the higher bandwidth 'FlexRay' within the next few years. Therefore interest begins to move towards FlexRay, rather than a further change to CAN.

6.7 Future Work

This thesis has provided a practical framework for using CAN in dependable systems. Nevertheless, as Section 6.5 explained, dependability is only achieved through the combination of many attributes. Work continues in this field including the following specific areas of interest.

A useful addition would be the ability to apply probabilistic analysis to multiple invocations. Although a technique was suggested in Chapter 4, the computational complexity of doing so is infeasible. Further investigation may lead to a way to incorporate probabilistic fault analysis with multiple invocation analysis.

There is considerable new work [17, 30, 106] on the use of flexible timing requirements in control systems. This is closely related to the use of non-harmonic periods for weakly-hard analysis. The link between these areas of work is yet to be made explicit.

Concerning the TCAN protocol. This thesis has presented TCAN in the context of a run-time guard against extreme faults, but there is another possible use for the protocol, as suggested in Chapter 5, that deserves further consideration. Neglecting faults, setting the delivery threshold to values less than the worst case response time, $X_i < R_i$, where there are non-harmonic periods, may lead to an effective way of providing a very high bus utilisation and a high reliability whilst maintaining weakly-hard constraints. This has similarities with EDF scheduling, but using the fixed priority mechanism directly rather than attempting to encode deadlines into the arbitration field.

Finally, the *babbling idiot failure* [160] (where a faulty node repeatedly suffers commission faults and therefore consumes more resources than it would normally use) has been a longstanding issue in real-time communication, although little research has been done. Preventing babbling idiots is a major issue in future dependable systems [19]. Some recent progress has been made on detection of the babbling idiot failure with CAN [32, 34].

6.8 Concluding Remarks

This thesis has contributed two forms of bus analysis to provide evidence of predictable behaviour at high levels of faults, it has re-addressed the standard fault model used for CAN analysis, suggested using non-harmonic periods where possible to increase fault-tolerance, and proposed the Timely-CAN protocol.

This thesis argues that through the use of these techniques, the flexibility of event-triggered communication may be exploited to provide a dependable communication mechanism for real-time systems.

List of References

[1] AEEC. Design guidance for integrated modular avionics. ARINC 651 (Draft 9), AEEC, Sept 1991.

[2] C. Almeida, J. Rufino, and P. Veríssimo. DDRAFT: Supporting dynamic distributed real-time applications with fault-tolerance. Technical Report CSTC RT-98-02, Centro de Sistemas Telemáticos e Computacionais do Instituto Superior Técnico, Lisboa, Portugal, Feb 1998.

[3] L. Almeida. *Flexibility and Timeliness in Fieldbus-based Real-time Systems*. PhD thesis, University of Aveiro, Portugal, Nov 1999.

[4] L. Almeida and J. Fonseca. FTT-CAN: A network-centric approach for CAN-based distributed systems. In *IFAC 4th Symposium on Intelligent Components and Instruments for Control Applications*, Buenos Aires, Argentina, Sept 2000.

[5] L. Almeida, P. Pedreiras, and J. A. Fonseca. FTTCAN: why and how? *IEEE Transactions on Industrial Electronics*, Jun 2002.

[6] R. Alur and D. Dill. Automata for modeling real-time systems. In M. S. Paterson, editor, *Proceedings of the 17th International Colloquium on Automata, Languages and Programming (ICALP)*, Warwick University, 1990.

[7] E. Anceaume and I. Puaut. A taxonomy of clock synchronization algorithms. Technical Report 1103, Institut de Recherche en Informatique et Systèmes Aléatoires, www.irisa.fr, Jul 1997.

[8] ARINC. *ARINC Specification 629 Multi-transmitter data bus*. Aeronautical Radio INC, 2551 Riva Road Annapolis, Maryland, 1990.

[9] ASSC. Guide to low and medium speed digital interface standards for avionic applications. Technical Report ASSC/110/2/42, Avionic Systems Standardisation Committee, Mar 1999.

[10] K. J. Åström and B. Wittenmark. *Computer Controlled Systems*. Prentice Hall, third edition, 1997. ISBN 0133148998.

[11] N. C. Audsley, A. Burns, M. Richardson, K. Tindell, and A. J. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Real-time Systems*, 8(5):284–292, 1993.

[12] N. C. Audsley, A. Burns, M. F. Richardson, and A. J. Wellings. Hard Real-Time Scheduling: The Deadline Monotonic Approach. In *Proceedings 8th IEEE Workshop on Real-Time Operating Systems and Software*, Atalanta, 1991.

[13] N. C. Audsley and A. Grigg. Timing analysis of the ARINC 629 databus for real-time applications. *Microprocessors and Microsystems*, 21:55–61, 1997.

[14] N. C. Audsley and A. J. Wellings. Analysing APEX applications. In *Proceddings of the 17th IEEE Real-time Systems Symposium*, pages 39–44, Washington, D.C., USA, 1996.

[15] J. Azevedo and N. Cravoisy. *The WorldFIP protocol*. WorldFIP International Technology Centre, Clamart, France, Aug 1998.

[16] I. Bate, J. McDermid, and P. Nightingale. Establishing timing requirements for control loops in real-time systems. *Microprocessors and Microsystems*, 27(4):159–169, 2003.

[17] I. Bate, P. Nightingale, and J. McDermid. Establishing timing requirements and control attributes for control loops in real-time systems. In *Proceedings of the 15th Euromicro Conference on Real-Time Systems*, pages 121–128, Jul 2003.

[18] I. J. Bate. *Scheduling and Timing Analysis for Safety Critical Real-Time Systems*. PhD thesis, Department of Computer Science, University of York, York, YO10 5DD, 1999.

[19] R. Belschner, J. Berwanger, C. Ebner, H. Eisele, S. Führer, T. Forest, T. Führer, F. Hartwich, B. Hedenetz, R. Hugel, A. Knapp, J. Krammer, A. Millsap, B. Müller, M. Peller, and A. Schedl. *FlexRay Requirements Specification*. FlexRay, www.flexray.com, v2.0.2 edition, Apr 2002.

[20] S. J. Berger. ARINC-629 digital communication system application on the 777 and beyond. In *ERA Avionics Conference*, 1995.

[21] G. Bernat. *Specification and Analysis of Weakly Hard Real-time Systems*. PhD thesis, Universitat de les Illes Balears, Palma, Spain, Jan 1998.

[22] G. Bernat. Weakly hard real-time systems. *IEEE Transactions on Computers*, 50(3):308–321, Mar 2001.

[23] G. Bernat and A. Burns. Weakly-hard temporal constraints. Tech. Report YCS-99-320, Department of Computer Science. University of York, 2000.

[24] G. Bernat, A. Burns, and A. Llamosi. Weakly hard real-time systems. *IEEE Transactions on Computers*, 50(4):308–321, Apr 2001.

[25] G. Bernat and R. Cayssials. Guaranteed on-line weakly-hard real-time systems. In *22nd IEEE Real-Time Systems Symposium*, pages 25–35, London, UK, Dec 2001.

[26] S. Biyabani, J. A. Stankovic, and K. Ramamritham. The integration of criticalness and deadlines in scheduling real-time tasks. In *Proceeding of the 9th IEEE Real-Time Systems Symposium*, pages 152–169, 1988.

[27] D. R. Boggs, J. C. Mogul, and C. A. Kent. Measured capacity of an ethernet: myths and reality. *Computer Communication Reviews*, 18(4):222–34, 1988.

[28] Bosch, Postfach 50, D-700 Stuttgart 1. *CAN Specification*, version 2.0 edition, 1991.

[29] I. Broster. Distributed real-time safety-critical control systems. Qualifying dissertation, Department of Computer Science, University of York, 2000.

[30] I. Broster, N. C. Audsley, G. Bernat, A. Burns, G. Buttazzo, P. Gai, M. González Harbour, and G. Lipari. First architecture framework: Key scheduling technologies for the future. Technical Report D-AF1-v1, University of York, 2003. FIRST EU Project deliverable.

[31] I. Broster, G. Bernat, and A. Burns. Weakly hard real-time constraints on controller area network. In *Proceedings of the 14th Euromicro Real-time Systems Conference*, pages 134–141, Vienna, Jun 2002.

[32] I. Broster and A. Burns. The babbling idiot in event-triggered real-time systems. In G. Fohler, editor, *Proceedings of the Work-In-Progress Session, 22nd IEEE Real-Time Systems Symposium, YCS 337*, pages 25–28. IEEE, Department of Computer Science, University of York, 2001.

[33] I. Broster and A. Burns. Timely use of the CAN protocol in critical hard real-time systems with faults. In *Proceedings of the 13th Euromicro Conference on Real-time Systems*, pages 95–102, Delft, The Netherlands, Jun 2001. IEEE.

[34] I. Broster and A. Burns. An analysable bus-guardian for event-triggered communication. In *Proceedings of the 24th Real-time Systems Symposium*, pages 410–419, Cancun, Mexico, Dec 2003. IEEE.

[35] I. Broster, A. Burns, and G. Rodríguez-Navas. Probabilistic analysis of CAN with faults. In *Proceedings of the 23rd Real-time Systems Symposium*, pages 269–278, Austin, Texas, 2002.

[36] M. J. Buckingham. *Noise in Electronic Devices and Systems*. Series in Electrical and Electronic Engineering. Ellis Horwood/Wiley, 1983.

[37] A. Burns. How to verify a safe real-time system: The application of model checking and timed automata to the production cell case study. *Real-Time Systems Journal*, 24(2):135–152, Mar 2003.

[38] A. Burns, S. Punnekkat, L. Strigini, and D. Wright. Probabilistic scheduling guarantees for fault-tolerant real-time systems. Technical Report YCS-311, Department of Computer Science, University of York, 1998.

[39] A. Burns and A. J. Wellings. *Real-Time Systems and Programming Languages*. Addison Wesley, 3rd edition, 2001.

[40] D. W. Caldwell and S. N. Chau. Spacecraft information systems. Technical report, NASA, 1993. Available from http://fst.ipl.nasa.gov/library/papers/avionix/.

[41] A. Carpenzano, R. Caponetto, L. Lo Bello, and O. Mirabella. Fuzzy traffic smoothing: an approach for real-time communication over ethernet networks. In *th IEEE International Workshop on Factory Communication Systems, WFCS?02*, Västerås, Sweden, Aug 2002. IEEE.

[42] W. C. Carter. A time for reflection. In *Proc. 8th IEEE Int. Symposium on Fault Tolerant Computing (FTCS-8)*, page 41, Santa Monica, Jun 1982.

[43] A. Cervin. Analyzing the effects of missed deadlines in control systems. In *ARTES Real-Time Graduate student conference*, pages 17–26, Lund, Sweden, Mar 2001.

[44] J. Charzinski. Performance of the error detection mechanisms in CAN. In *Proceedings of the 1st International CAN Conference*, pages 20–29, Mainz, Sept 1994.

[45] CiA. *CANopen*. CiA (CAN in Automation), 2002. EN 50325-4 Standard.

[46] F. Cristian and C. Fetzer. The timed asynchronous distributed system model. *IEEE Transactions on Parallel and Distributed Systems*, 10(6):642–657, 1999.

[47] Echelon. *Introduction to the LONWORKS System*. Echelon Corporation, 4015 Miranda Avenue, Palo Alto, CA 94304, USA, 1.0 edition, 1999.

[48] C. A. Ericson. Software and system safety. In *Proceedings of the 5th International Safety Conference*, volume 1-1, pages III–B–1–III–B–11. System Safety Soc., 1981.

[49] David Evans. EMI: A serial killer? *Aviation Today: Avionics Magazine, PBI Media*, Nov 2000. Archive available from http://www.aviationtoday.com/reports/avionics/previous/1100/column3.htm.

[50] P. Ferriol, F. Navio, J. Pons, J. Proenza, and J. Miro-Julia. A double CAN architecture for fault-tolerant control systems. In *5th International CAN Conference, ICC'98*, San Jose CA, Nov 1998.

[51] L.-B. Fredriksson. A CAN kingdom. Technical report, KVASER AB, Sweden, 1995.

[52] T. Führer, B. Muller, W. Dieterle, F. Hartwich, R. Hugel, and M. Walther. Time triggered communication on CAN. Technical report, Robert Bosch GmbH, 2000. Available from http://www.can.bosch.com/.

[53] P. Gai, L. Abeni, M. Giorgi, and G. Buttazzo. A new kernel approach for modular real-time systems development. In *Proceedings Euromicro Conference on Real-time Systems*, pages 199–206, Delft, The Netherlands, Jun 2001. IEEE.

[54] B. Gaujal and N. Navet. Fault confinement mechanisms on CAN: Analysis and improvements. In *Proc. 4th FeT IFAC Conference, FeT'2001, Fieldbus Technology*, Nancy, France, Nov 2001.

[55] B. Gaujal and N. Navet. Fault confinement mechanisms of the CAN protocol: Analysis and improvements. Rapport de Recherche RR-4603, Institut National De Recherche en Informatique et en Automatique, France, Oct 2002.

[56] M. Gergeleit and H. Steich. Implementing a distributed high-resolution real-time clock using the CAN bus. In *Proceedings of the 1st International CAN Conference*. CiA, 1994.

[57] G.Rodríguez-Navas, M. Barranco, J. Proenza, and I. Broster. COTS-based hardware support to timeliness in CAN networks. In *Emerging Technologies and Factory Automation*, Lisbon, Portugal, Sept 2003. (To Appear).

[58] J. Y. Halpern and Y. Moses. Knowledge and common knowledge in a distributed environment. *Journal of the ACM*, 37(3):549–587, Jul 1990.

[59] R. W. Hamming. Error detecting and error correcting codes. *The Bell System Tech. Journal*, XXIX(2):147–160, 1950.

[60] P. Hansen. FlexRay protocol picks up support. *The Hansen Report on Automotive Electronics*, 15(2), Jun 2002. www.hansenreport.com.

[61] F. Hartwich, B. Muller, T. Führer, and R. Hugel. CAN network with time-triggered communication. In *7th international CAN Conference*. CAN in Automation GmbH, Oct 2000.

[62] S. Heathfield. The ARINC-629 databus. DCSC report DCSC/TN/92/30, University of York, Department of Computer Science, Dependable Computing Systems Centre, Oct 1992.

[63] Albert Helfrick. Avionics & portable electronics : Trouble in the air? *Avionics News Magazine*, Sept 1996.

[64] H. Hilmer, H.-D. Kochs, and E. Dittmar. A fault-tolerant communication architecture for real-time control systems. In *Proc. IEEE Int. Workshop on Factory Communication Systems*, Barcelona, Spain, Oct 1997.

[65] A. L. Hopkins, T. B. Smith, and J. H. Lala. FTMP—a highly reliable fault-tolerant multiprocessor for aircraft. *Proceedings of the IEEE*, 66(10):1221–39, Oct 1978.

[66] P. Horowitz and W. Hill. *The Art of Electronics*. Cambridge University Press, 2nd edition, Sept 1989. ISBN 0521377099.

[67] R. A. Hulsebos. The fieldbus reference list. Available from http://ourworld.cs.com/rahulsebos/Links.htm, (Referenced)2002.

[68] IBUS. *The INTERBUS System—INTERBUS UART's view*. INTERBUS Support Center, Jun 1998.

[69] IEE. EMC and functional safety. IEE guidance document, IEE, Sept 2000. Available from http://www.iee.org.uk/PAB/EMC/core.htm.

[70] IEEE. *802.3 Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications*. IEEE Computer Society, 3 Park Avenue, New York, NY 10016-5997, USA, Mar 2002.

[71] Intel. *Automotive Products*. Number 281792-007 in Automotive Series. Intel Literature, 1994.

[72] International Standards Organisation. *ISO 11898. Road Vehicles—Interchange of digital information—Controller area network (CAN) for high speed communication*, 1993.

[73] ISO. *Information technology—Open Systems Interconnection—Basic Reference Model: The Basic Model*. ISO/IEC, 7498-1 edition, 1994.

[74] ISO. Road vehicles—low-speed serial data communication—part 2: Low-speed controller area network (CAN). Technical Report 11519-2, International Organization for Standardization, 1994.

[75] ISO. *Road Vehicles—Interchange of digital information—Controller area network (CAN) part 4: Time triggered Communication*. International Standards Organisation, 2000. Working Draft.

[76] F. Jahanian and A. Mok. Safety analysis of timing properties in realtime systems. *IEEE Trans. Software Eng*, SE-12(9):890–904, Sept 1986.

[77] D. M. Johnson. Integrated modular avionics. *Computer systems scientific and engineering*, 11(3):125–133, May 1996.

[78] J. Kaiser and M. A. Livani. Invocation of real-time objects in a CAN-Bus based system. In *Proceedings of the First International Symposium on Object Oriented Distributed Real-time Computing Systems*, Kyoto, Apr 1998.

[79] J. Kaiser and M. A. Livani. Achieving fault-tolerant ordered broadcasts in CAN. In *European Dependable Computing Conference*, pages 351–363, 1999.

[80] H. Kim and K. G. Shin. Modeling of externally-induced/common-cause faults in fault-tolerant systems. Technical report, Real-time Computing Lab, Depart-

ment of Electrical Engineering and Computer Science, University of Michigan, 1993.

[81] H. Kopetz. A solution to an automotive control system benchmark. In *Proc. 15th IEEE Real-Time Systems Symposium*, pages 154–158, Puerto Rico, Dec 1994. IEEE.

[82] H. Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Kluwer Academic, 1997.

[83] H. Kopetz. Time-triggered model of computation. In *Proceedings 19th Real-Time Systems Symposium*, pages 168–177, Madrid, Spain, Dec 1998.

[84] H. Kopetz and W. Ochsenreiter. Clock synchronisation in distributed real-time systems. *IEEE Trans. Computers*, 36(8):933–940, 1987.

[85] Seok-Kyu Kweon, Kang G. Shin, and Qin Zheng. Statistical real-time communication over ethernet for manufacturing automation systems. In *IEEE Real Time Technology and Applications Symposium*, pages 192–202, 1999.

[86] Peter B. Ladkin. Electromagnetic interference with aircraft systems: why worry? Technical Report RVS-J-97-03, University of Bielefeld—Faculty of Technology, 1997.

[87] G. Le Lann. Deterministic multiple access protocols for real-time local area networks. Research Report 246, INRIA, France, 1983.

[88] J. C. Laprie. *Dependability—Basic Concepts and Terminology*, volume 5 of *Dependable Computing and Fault-tolerant Systems*. Springer-Verlag, 1992. IFIP WG 10.4.

[89] G. Leen and D. Heffernan. Expanding automotive electronic systems. *IEEE Computer*, 0018-9162/02:88–93, Jan 2002.

[90] J. P. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *Proceedings of the 10h IEEE Real-Time Systems Symposium*, pages 166–171, Santa Monica, California, USA, Dec 1989. IEEE, Computer Society Press.

[91] J. Y. T. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic real-time tasks. *Performance Evaluation*, 2(4):237–250, Dec 1982.

[92] N. G. Leveson. Software safety: why, what and how. *ACM Computing Surveys*, 18(2):125–163, 1986.

[93] N. G. Leveson. *Safeware: system safety and computers*. Addison-Wesley, 1995. ISBN 0201119722.

[94] Bev Littlewood and David Wright. Some conservative stopping rules for the operational testing of safety-critical software. *Software Engineering*, 23(11):673–683, 1997.

[95] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

[96] M. A. Livani and J. Kaiser. EDF consensus on CAN bus access for dynamic real-time applications. In *IPPS/SPDP Workshops*, pages 1088–1097, 1998.

[97] M. A. Livani, J. Kaiser, and W. J. Jia. Scheduling hard and soft real-time communication in the controller area network. In *Proceedings of the 23rd IFAC/IFIP Workshop an Real-Time Programming*, 1998.

[98] M.A. Livani. SHARE: A transparent approach to fault-tolerant broadcast in CAN. In *Proceedings of the 6th International CAN Conference (ICC6)*, Torino, Italy, Nov 1999.

[99] C. D. Locke. *Best Effort decision Making for real-time scheduling*. PhD thesis, CMU, 1986. CMU-CS-86-134.

[100] H. Lonn and J. Axelsson. A comparison of fixed-priority and static cyclic scheduling for distributed automotive control applications. In *Proceedings of the 11th Euromicro Conference on Real-time Systems*, pages 142–149, York, England, UK, Jun 1999. IEEE Computer Society Press.

[101] G. A. Maciag. The next generation of insurance road warriors has hit the information superhighway. News item, Acord

Global Insurance Standards, 1999. Archive available from http://www.acord.org/News/NewsDetail.aspx?Type=1&aid=646.

[102] N. Malcolm and W. Zhao. Hard real-time communication in multiple access networks. *Real-time Systems*, 8(1):35–78, 1995. Kluwer.

[103] G. K. Manacher. Production and stabilization of real-time task schedules. *Journal of the ACM (JACM)*, 14(3):439–465, Jul 1965.

[104] P. Martí, J. M. Fuertes, and G. Fohler. Minimising sampling jitter degradation in real-time control systems. In *IV JORNADAS DE TIEMPO REAL*, Zaragoza. Spain, Feb 2001.

[105] P. Martí, J. M. Fuertes, G. Fohler, and K. Ramamritham. Jitter compensation for real-time control systems. In *Proceedings Real-Time Systems Symposium*, pages 39–48, London, UK, Dec 2001. IEEE.

[106] Pau Martí, Josep M. Fuertes, Gerhard Fohler, and Krithi Ramamritham. Improving quality-of-control using flexible timing constraint : Metric and scheduling issues. In *Proceedings Real-Time Systems Symposium*, pages 91–100, Austin, Texas, Dec 2002. IEEE.

[107] R. T. McLaughlin. EMC susceptibility testing of a CAN car. SAE Technical Paper 932886, SAE, 1993. Available from: http://www.warwick.ac.uk/devicenet/publications.htm.

[108] MoD. Safety management requirements for defence systems. Defence Standard 00-56, Ministry of Defence, Dec 1996.

[109] Motorola. *MC68336/376 Users Manual*. Motorola, 1996.

[110] C. J. Murray. Automotive groups divide on road to data bus. *EETimes*, Sept 2001. Number OEG20010927S0080. Archive available from http://www.eetimes.com/story/OEG20010927S0080.

[111] C. J. Murray. Automakers asked to can CAN. *EETimes*, May 2002. Number OEG20020531S0078. Archive available from http://www.eetimes.com/story/OEG20020531S0078.

[112] M. Natale. Scheduling the CAN bus with earliest deadline techniques. In *Proceedings of the 21st IEEE Real-time Systems Symposium*, pages 259–268, Florida, USA, Dec 2000. IEEE.

[113] N. Navet and Y.-Q. Song. Validation of real-time in-vehicle applications. *Computers in Industry*, 46(2):107–122, 2001.

[114] 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*, 46(1):607–617, 2000.

[115] NHTSA. Development, evaluation, and demonstration of a tractor trailer intelligent communication and power link. Technical Report DOT HS 808685, NHTSA, Jan 1998.

[116] J. Nilsson, B. Bernhardsson, and B. Wittenmark. Some topics in real-time control. In *Proceedings of the 17th American Control Conference*, pages 2386–2390, Philadelphia, Pennsylvania, Jun 1998.

[117] T. Nolte. *Reducing Pessimism and Increasing Flexibility in the Controller Area Network.* Licentiate thesis, Mälardalens University, Västerås, Sweden, 2003.

[118] T. Nolte, H. Hansson, C. Norström, and S. Punnekkat. Using bit-stuffing distributions in CAN analysis. In *IEEE Real-Time Embedded Systems Workshop at the Real-Time Systems Symposium*, London, UK, 2001.

[119] T. Nolte, H. Hansson, and M. Sjödin. Efficient and fair scheduling of periodic and aperiodic messages on CAN using EDF and constant bandwidth servers. MRTC Report ISSN 1404-3041 ISRN MDH-MRTC-73/2002-1-SE, Mälardalens University, Mälardalen Real-Time Research Centre, Mälardalen University, May 2002. Available from http://www.mrtc.mdh.se/showPublications.phtml.

[120] B. Patt. A theory of clock synchronization. Technical Report MIT/LCS/TR-680, Institut de Recherche en Informatique et Systèmes Aléatoires, France, 1994.

[121] C. R. Paul. *Introduction to electromagnetic compatibility*. Wiley, New York, 1992. ISBN0471549274.

[122] P. Pedreiras, L. Almeida, and P. Gai. The FTT-Ethernet protocol: Merging flexibility, timeliness and efficiency. In *Proceedings of the 14th Euromicro Conference on Real-Time Systems (ECRTS'02)*, pages 152–160, Vienna, Austria, Jun 2002. IEEE, Computer Society Press.

[123] M. Peller, J. Berwanger, and R. Grießbach. *byteflight specification*. BWM AG, draft edition, Nov 1999. Available From www.byteflight.com.

[124] Philips. SJA1000 stand-alone CAN controller. Technical Report IC18 SJA1000, Philips Semiconductors Ltd, Jan 2000.

[125] Philips. Tja1054; fault-tolerant can transceiver. Datasheet IC18/TJA1054, Philips Semiconductors, 2000.

[126] L. M. Pinho and F. Vasques. Timing analysis of reliable real-time communication in CAN networks. In *Proceedings of the 13th Euromicro Conference on Real-time Systems*, Delft, The Netherlands, 2001.

[127] L. M. Pinho, F. Vasques, and E. Tovar. Integrating inaccessibility in response time analysis of CAN networks. In *Proceedings of the 3rd IEEE Workshop On Factory Communication Systems*, pages 77–84, Porto, Portugal, Sep 2000.

[128] T. Pop, P. Eles, and Z. Peng. Holistic scheduling and analysis of mixed time/event-triggered distributed embedded systems. In *Proceedings of the 10th International Symposium on Hardware/Software Codesign (CODES 2002)*, pages 187–192, Estes Park, Colorado, USA, May 2002.

[129] J. Proenza and J. Miro-Julia. MajorCAN: A modification to the Controller Area Network protocol to achieve Atomic Broadcast. In *IEEE Int. Workshop on Group Communications and Computations (IWGCC)*, Taipei, Taiwan, Apr 2000.

[130] ProfiBus. *Technology and Application—System Description*. ProfiBus International Support Center, Haid-und-Nau-Straße 7, D-76131 Karlsruhe, Germany, Oct 2002.

[131] S. Punnekkat, H. Hansson, and C. Norström. Response time analysis under errors for CAN. In *Proceedings of the 6th Real-Time Technology and Applications Symposium (RTAS)*, pages 258–265, Washington DC, 2000. IEEE.

[132] M. Rahnema. Overview of the GSM system and protocol architecture. *IEEE Communications Magazine*, 31(3):92–100, Apr 1993.

[133] J. H. Reppy. *Concurrent Programming with Events—The Concurrent ML Manual*. AT&T Bell Lab., version 0.9.8 edition, Feb 1993.

[134] P. Richards. Timing properties of multiprocessor systems. Technical Report TD-B60-27, Tech. Operations, Inc., Burlington, Mass., Aug 1960.

[135] H. Rischel and H. Sun. Design and prototyping of real-time systems using CSP and CML. In *Proceedings of the 9th Euromicro Workshop on Real Time Systems*, pages 121–129, Toledo, Spain, Jun 1997.

[136] M. A. Rivas and M. G. Harbour. POSIX-compatible application-defined scheduling in MaRTE OS. In *Proceedings of the 14h Euromicro Conference on Real-time Systems*, pages 67–75, Vienna, Austria, Jun 2002. IEEE.

[137] L. Rodrigues, M. Guimarães, and J. Rufino. Fault-tolerant clock synchronization in CAN. In *Proceedings of the 19th Real-Time Systems Symposium*, pages 420–429, Madrid, Spain, Dec 1998. IEEE.

[138] Guillermo Rodríguez-Navas, Manuel Barranco, Julián Proenza, and Ian Broster. COTS-based hardware support to timeliness in CAN networks. In *Proceedings of the 9th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA2003)*, Lisbon, Portugal, Sept 2003. IEEE.

[139] M. Rucks. Optical layer for CAN. In *Proceeding of the 1st International CAN Conference*, volume 2, pages 11–18, Mainz, Germany, 1994. CiA.

[140] J. Rufino. *Computational System for Real-time Distributed Control*. PhD thesis, Universidade Técnica de Lisboa Instito Superior Técnico, Jul 2002.

[141] J. Rufino and P. Veríssimo. Hard real-time operation of CAN. Technical Report CSTC RT-97-02, Centro de Sistemas Telemáticos e Computacionais, Instituto Superior Técnico, Lisboa, Portugal, Jan 1997.

[142] J. Rufino, P. Veríssimo, and G. Arroz. Defining a CAN-based infrastructure for fault-tolerant real-time distributed computing. In *Proceedings of the 19th Real-Time Systems Symposium*, Work In Progress, pages 27–30, Madrid, Spain, Dec 1998. IEEE. (published as Technical Report UNL-CSE-98-002, from University of Nebraska-Lincoln, Department of Computer Science and Engineering).

[143] J. Rufino, P. Veríssimo, and G. Arroz. A Columbus' egg idea for CAN media redundancy. In *Digest of Papers, The 29th International Symposium on Fault-Tolerant Computing Systems*, pages 286–293, Madison, Wisconsin, USA, Jun 1999. IEEE.

[144] J. Rufino, P. Veríssimo, and G. Arroz. Embedded platforms for distributed real-time computing: Challenges and results. In *Proceedings of the 2nd International Symposium on Object-oriented Real-time distributed Computing*, pages 147–152, Saint Malo, France, May 1999. IEEE.

[145] J. Rufino, P. Veríssimo, G. Arroz, C. Almeida, and L. Rodrigues. Fault-tolerant broadcasts in CAN. In *Digest of Papers, The 28th International Symposium on Fault-Tolerant Computing Systems*, pages 150–159, Munich, Germany, Jun 1998. IEEE.

[146] J. Rushby. A comparison of bus architectures for safety-critical embedded systems. Technical report, Computer Science Laboratory, SRI International, Menlo Park, CA, 2001.

[147] SAE. Class C application requirement considerations. Technical Report J2056/1, Society of Automotive Engineers, 1993.

[148] SAE. *Single Wire J2411: CAN Network for Vehicle Applications*. SAE, Available from www.sae.org, Feb 2000.

[149] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423 and 623–656, Jul and Oct 1948.

[150] N. J. A. Sloane. Sequence a000108—catalan numbers. Available from http://www.research.att.com/~njas/sequences/Seis.html, 2003. On-Line Encyclopedia of Integer Sequences.

[151] T. Standage. *The Turk: The Life and Times of the Famous 18th Century Chess Playing Machine*. Walker and Co, Apr 2002. ISBN 0802713912.

[152] D. L Stanislaw. ARINC-629: the new airline databus. *Avionics*, Jun, Jul, Aug 1989.

[153] J. A. Stankovic, A. Burns, K. Jeffay, M. Jones, G. Koob, I. Lee, J. Lehoczky, J. Liu, A. Mok, K. Ramamritham, J. Ready, L. Sha, and A. van Tilborg. Strategic directions in real-time and embedded systems. *ACM Computing Surveys*, 28(4):751–763, 1996.

[154] N. Suri, C. J. Walter, and M. M. Hugue. *Advances In Ultra-Dependable Distributed Systems*, chapter 1. IEEE Computer Society Press, 10662 Los Vaqueros Circle, P.O. Box 3014, Los Alamitos, CA 90720-1264, 1995.

[155] J. P. Thomesse. A review of the fieldbuses. *Annual Reviews in Control*, 22:35–45, 1998.

[156] H. A. Thompson, H. Benitez-Perez, D. Lee, D. N. Ramos-Hernandez, P. J. Fleming, and C. G. Legge. A CANbus-based safety-critical distributed aero-engine control systems architecture demonstrator. *Microprocessors and Microsystems*, 23:345–355, 1999.

[157] K. Tindell. *Fixed Priority Scheduling of Hard Real-Time Systems*. PhD thesis, Department of Computer Science, University of York, 1994.

[158] K. Tindell and A. Burns. Guaranteed message latencies for distributed safety-critical hard real-time networks. Technical Report YCS 229, Department of Computer Science, University of York, May 1994.

[159] K. Tindell, A. Burns, and A. J. Wellings. Calculating controller area network (CAN) message response times. *Control Engineering Practice*, 3(8):1163–1169, 1995.

[160] K. Tindell and H. Hansson. Babbling idiots, the dual-priority protocol, and smart CAN controllers. In *Proceedings of the 2nd International CAN Conference*, pages 7.22–28, 1995.

[161] M. R. Tolhurst. *Open Systems Interconnection*. Macmillan Computer Science Series. Macmillan Education, 1988.

[162] Eduardo Manuel de Medicis Tovar. *Supporting Real-Time Communications with Standard Factory-Floor Networks*. PhD thesis, Universidade do Porto, 1999.

[163] TTP. TTP/C protocol specification. Technical report, TTTech Computertechnik, Wien, Austria, Jul 1999.

[164] TTTech. TTP plan—the TTP cluster design tool. Technical report, TTTech Computertechnik AG, 2002. Available from http://www.tttech.com/.

[165] Klaus Turski. A global time system for CAN networks. In *Proceedings of the 1st International CAN Conference*. CiA, 1994.

[166] J. Unruh, H. J. Mathony, and K. H. Kaiser. Error detection capabilities of the CAN protocol. Technical report, Robert Bosch GmbH, Stuttgart, Germany, Dec. 1989.

[167] P. Veríssimo, J. Rufino, and L. Ming. How hard is hard real-time communication on field-buses? In *Digest of Papers, The 27th International Symposium on Fault-Tolerant Computing Systems*, pages 112–121, Seatle, Washington, USA, Jun 1997. IEEE.

[168] H. E. Waterman. FAA's certification position on advanced avionics. *AIAA Astronaut. Aeronaut*, pages 49–51, May 1978.

[169] John Woods. News article on EMI affecting Black Hawk helicopter. *Risks Digest*, 5(56), Nov 1987. Archive available from http://catless.ncl.ac.uk/Risks.

[170] H. Zeltwanger. Failure detection and error handling in CAN-based networks. In *Seminario Anual de Automática, Electrónice Industrial e Instrumentación*, Pamplona, Spain, Sept 1998.

[171] K. M. Zuberi and K. G. Shin. Non-preemptive scheduling of messages on controller area network for real-time control applications. In *Proceedings of Real-Time Technology and Applications Symposium*, pages 240–259, Chicago, Illinois, May 1995.

[172] K. M. Zuberi and K. G. Shin. Scheduling messages on controller area network for real-time CIM applications. *IEEE Transactions on Robotics and Automation*, 13(2):310–314, 1997.