Share-Driven Scheduling of Embedded Networks

Thomas Nolte

May 2006
Abstract

Many products are built from more or less independently developed subsystems. For instance, a car consists of subsystems for transmission, braking, suspension, etc. These subsystems are frequently controlled by an embedded computer system. In the automotive industry, as well as in other application domains, there is currently a trend from an approach where subsystems have dedicated computer hardware and other resources (a federated approach) to an approach where subsystems share hardware and other resources (an integrated approach). This is motivated by a strong pressure to reduce product cost, at the same time as an increasing number of subsystems are being introduced.

When integrating subsystems, it is desirable that guarantees valid before integration are also valid after integration, since this would eliminate the need for costly reverifications. The computer network is a resource that is typically shared among all subsystems. Hence, a central issue when integrating subsystems is to provide an efficient scheduling of message transmissions on the network. There are essentially three families of schedulers that can be used: priority-driven schedulers that assign priorities to messages, time-driven schedulers that assign specific time-slots for transmission of specific messages, and share-driven schedulers that assign shares of the available network capacity to groups of messages.

This thesis presents a framework for share-driven scheduling, to be implemented and used in embedded networks, with the aim to facilitate subsystem integration by reducing the risk of interference between subsystems. The framework is applied in the automotive domain.

The initial parts of the thesis give an overview of systems, subsystems and network technologies found and used in the automotive domain. Then, the share-driven scheduling framework is presented, analytically investigated and proven, as well as evaluated in a simulation study. Finally it is shown how the framework is to be configured and used in the context of subsystem integration. The results show that the framework allows for flexible and efficient scheduling of messages with real-time constraints, facilitating integration of subsystems from a network point of view.

Keywords: subsystem integration, controller area network, CAN, real-time communication, real-time analysis, server-based scheduling, Server-CAN
To Karin and Jürgen
Acknowledgements

The journey that has resulted in, among other things, this thesis started on a hot summer day in Västerås, back in 2000. I was currently doing my Bachelors degree project at Adtranz (now Bombardier Transportation). Going on my daily lunch down-town, I ran into my supervisor at the time, Christer Norström. During lunch together he presented the idea of a continuation of my studies towards a Ph.D. degree. At that time the thought of postgraduate studies had never crossed my mind, but since that day, almost 6 years ago, I know that I could not have made a better choice. Thank you Christer, for giving me this opportunity.

The work presented in this thesis would not have been possible without the help of my supervisors Hans Hansson, Christer Norström and Sasikumar Punnekkat. You have believed in me, supported my work, fed me with interesting tasks (both related and unrelated to this thesis) and inspired me throughout these years. I am very fortunate to have had such a good crew as supervisors!

Thank you Mikael Nolin for all the work that we have done together these past years, and all the nice travel time spent in various places. I wrote your name among my supervisors above before I realised that you actually never officially were my supervisor, but it feels as if you were.

Looking at the actual thesis work (the 6 months of writing spent resulting in this thesis), thank you Iain Bate who gave good constructive suggestions for the structure and style of my thesis based on an early version in Miami last year. Thank you Lucia Lo Bello for, apart from being a great friend, taking me in as a visiting researcher even though my thesis writing was not yet finished. My time in Italy allowed me to think and form the thesis in a way that would have been much harder back in Sweden. Thank you Niklas Nolte (my brother) for running my simulations these past months so that I could use my own computer to write the thesis. Also, a big thank you goes to my proof reading squad who helped in improving the quality of the thesis: Hans Hansson,
Lesson learned: Writing a thesis was more work than I expected. The road towards this thesis included research work together with several local and international colleagues. Thank you for having the time and interest leading to good publications over the past years. Thank you Hans Hanson, Christer Norström, Sasikumar Punnekkat, Mikael Nolin, Lucia Lo Bello, Guillermo Rodríguez-Navas, Julián Proenza, Dag Nyström, Anders Möller, Mikael Åkerholm, Kwei-Jay Lin, Iain Bate, Stefan M. Petters, David Andrews and Clara Maria Otero Pérez. You are all somewhat responsible for my nice travel these past years :-)

A big thank you goes to Dag Nyström, who (what I know and many suspect) still not has got over our separation in late 2004. Before our separation we used to share more time than we shared with our respective families. The story goes back to 1997 when we both, with very different backgrounds, started our undergraduate education, ending up studying daily together for 4 years, starting as graduate students together, and now, after a total of 8.5 years (yes, only 8 years for you Dag and 8.5 years for me), we both got our Ph.D.

During my everyday life at the department, some people have shared my projects and ideas more than others. Thank you Daniel Sundmark for being a great friend, ambitious co-worker, great chef, and an average runner. You will make it far. Thank you Anders Pettersson, for always being there and finding the time to make our projects successful. We both had our first official research presentations at the same conference back in 2001. You should use less salt though. Thank you Johan Fredriksson for being not just a colleague, but a personal trainer, travel mate during holidays and a great friend! Thank you Markus Lindgren for all nice talks, dinners and discussions on (necessary) gadgets. However, I think you could be a bit more serious in your training.

Some of us graduate students shared time also during the undergraduate education. We have over these past years had endless discussions both during work hours and off work hours, both in Sweden and in (sometimes) exotic countries. Thank you for the great time we have shared (the old crew starting 1997) Dag Nyström, Joel Huselius, Markus Nilsson and Larisa Rizvanović, and the newcomers starting 1998: Johan Fredriksson, Jonas Neander, Johan Andersson, Mikael Åkerholm and Kaj Hänninen.

When I started as a graduate student at the department back in (let’s say) November 2000, soon I got to know many colleagues that inspired me to do good and hard work the upcoming years. Thank you for the great time we
spent these years during various kinds of projects, meetings, travel, both with and without beers. Thank you Markus Lindgren, Kristian Sandström, Anders Wall, Jukka Mäki-Turja, Mohammed El Shobaki, Filip Sebek, Gerhard Fohler, Damir Isović, Tomas Lennvall, Radu Dobrin and Henrik Thane.

During my two years as the university’s Ph.D. student representative, I got to know many colleagues working in other areas. We shared, apart from a genuine interest in the quality of our graduate education, many nice discussions and moments, often off work hours. Thank you Carina Andersson, Andreas Ryve, Emma Nehrenheim, Daniel Sundmark, Helena Jerregård, Helén Larsericsdotter, Andreas Backlund, Madelene Nilsson, Markus Nilsson, Björn Widarsson, Tommy Gunnarsson, Agneta Brav, Maria Dahlin, Marie Öhman, Gerrit Berends, Helena Stärner, Lasse Nurkkala, Pia Lindberg, Fredrik Wallin, Nona Sherif and more. Keep up the good work!

Apart from my colleagues mentioned above, there are several colleagues at the department that I have shared interesting projects, trips and time with over the years. Thank you Mats Björkman, Markus Bohlin, Ylva Bäcklund, Ivica Crnkovic, Malin Ekholm, Harriet Ekwall, Sigrid Eldh, Ralf Elvsén, Andreas Ermedahl, Daniel Flemström, Joakim Fröberg, Ewa Hansen, Roger Hassel, Ann Huget, Andreas Johnsson, Roger Jonsson, Annika Jonsson, Rikard Land, Magnus Larsson, Thomas Larsson, Maria Lindén, Lennart Lindh, Åsa Lundkvist, Frank Lüders, Bob Melander, Goran Mustapic, Pengpeng Ni, Christer Sandberg, Else-Maj Silén, Johan Stärner, Monica Wasell, Gunnar Widforss, Jan Osterberg and more. You have all contributed to making these past years a joy for me.

The path towards this thesis has been a pleasure in the variety of interesting work that had to be done as well as interesting conferences that had to be attended. Over the world, I have over these past 5 years shared many nice moments with a lot of national and international colleagues. I would like to thank some of you a bit extra, as when I think of you, I recall those nice moments. Thank you Luís Almeida, Mehdi Amirijoo, Anita Andler, Sten Andler, Neil Audsley, Manuel Barranco, Sanjoy Baruah, Iain Bate, Enrico Bini, Ian Broster, Alan Burns, Giorgio Buttazzo, Radu Dobrin, Mohammed El Shobaki, Tullio Facchinetti, Hugo Flordal, Johan Fredriksson, Joakim Fröberg, Gerhard Fohler, Shelby Funk, Paolo Gai, Hans Hansson, Jörgen Hansson, Hoai Hoang, Joel Huselius, Damir Isović, Giordano Kaczyński, Kwei-Jay Lin, Tomas Lennvall, Federica Lo Bello, Lucia Lo Bello, Pau Martí, Jukka Mäki-Turja, Daniel Mosse, Anders Möller, Jonas Neander, Mikael Nolin, Christer Norström, Dag Nyström, Carlos Eduardo Pereira, Clara Maria Otero Pérez, Stefan M. Petters, Julián Proenza, Ola Redell, Guillermo Rodríguez-Nava,
Finally, and next to most important, I thank my family and my friends for supporting me through these years, and taking the time to celebrate things that are important to me. Especially, I thank the southern part of my Swedish family who despite the geographical distance of (almost ;-) living in another country, still manages to find time to and be with me for all important events in my life. Also, I thank the Italian part of my family for all nice time that we have spent together, allowing me to recharge between heavy periods of work.

For last, and most important, I thank Carla for her love, for putting up with me even though I sometimes spend a lot of time at work or in travel, and, of course, for providing delicious food all these years. Taking it all together it is a powerful combination of fuel that can bring me far in life. I am fortunate.

This work has been supported by the Swedish Foundation for Strategic Research (SSF) via the research programmes ARTES and SAVE, the Knowledge Foundation (KK-stiftelsen) via the research profile grant Mälardalen Real-Time Research Centre (MRTC), LM Ericsson’s Research Foundation, and Mälardalen University.

Thomas Nolte, Catania, April 2006.
Notes for the reader

The topic of this thesis is share-driven scheduling of embedded networks. A share-driven scheduling framework is presented in chapters 6 to 9. Moreover, in Chapter 9 it is investigated how share-driven scheduling of embedded networks can be used to facilitate subsystem integration from a network point of view. These chapters are presenting the research contributions of this thesis. However, several of the other chapters in the thesis are of general nature and can be read separately.

For the reader interested in an introduction to real-time systems, real-time communications and real-time analysis, Chapter 2 gives a good overview of these topics as well as many references for further reading.

Anyone interested in Electronic Automotive Systems (EASs) may read Chapter 3, which investigates EASs in detail, presenting information on communications requirements, typical functions and subsystems used, as well as some real example architectures.

To know more about which network technologies that are used in the automotive domain (and for what), Chapter 4 is surveying the current state-of-the-art. Also, a brief overview of network technologies used in other application domains is given. A thorough description of CAN is given in Chapter 5.

To read about subsystem integration, readers are referred to Chapter 1, Chapter 3, Chapter 9 and Chapter 10.
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acknowledgements</td>
<td>vii</td>
</tr>
<tr>
<td>Notes for the reader</td>
<td>xi</td>
</tr>
<tr>
<td>Contents</td>
<td>xiii</td>
</tr>
<tr>
<td>List of publications (short)</td>
<td>xvii</td>
</tr>
<tr>
<td>Swedish summary</td>
<td>xxi</td>
</tr>
<tr>
<td>1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Subsystem integration</td>
<td>2</td>
</tr>
<tr>
<td>1.2 Problem description</td>
<td>3</td>
</tr>
<tr>
<td>1.3 Proposition and research questions</td>
<td>5</td>
</tr>
<tr>
<td>1.4 Research work and method</td>
<td>6</td>
</tr>
<tr>
<td>1.5 Contributions</td>
<td>7</td>
</tr>
<tr>
<td>1.6 Thesis outline</td>
<td>8</td>
</tr>
<tr>
<td>2 Theoretical background</td>
<td>11</td>
</tr>
<tr>
<td>2.1 Real-time systems</td>
<td>11</td>
</tr>
<tr>
<td>2.2 Real-time communications</td>
<td>16</td>
</tr>
<tr>
<td>2.3 Real-time scheduling</td>
<td>28</td>
</tr>
<tr>
<td>2.4 Real-time analysis</td>
<td>37</td>
</tr>
<tr>
<td>2.5 Summary</td>
<td>43</td>
</tr>
<tr>
<td>3 Electronic automotive systems</td>
<td>45</td>
</tr>
<tr>
<td>3.1 Historical perspective</td>
<td>47</td>
</tr>
<tr>
<td>3.2 Current trends</td>
<td>48</td>
</tr>
</tbody>
</table>

xiii
3.3 Communication requirements ........................................ 48
3.4 Typical subsystems ............................................... 51
3.5 Example of automotive systems ................................. 57
3.6 Summary ...................................................... 61

4 Embedded networks .................................................. 63
4.1 Automotive domain ............................................... 63
4.2 Other domains .................................................. 82
4.3 Summary ...................................................... 85

5 The Controller Area Network (CAN) ............................. 87
5.1 History ........................................................ 87
5.2 Applications .................................................. 90
5.3 Technical properties ............................................ 90
5.4 Scheduling of CAN ............................................. 91
5.5 Schedulability analysis .......................................... 92
5.6 Higher layer protocols ....................................... 97
5.7 Summary ...................................................... 102

6 Share-driven scheduling of CAN .................................. 105
6.1 Scheduler design ............................................... 106
6.2 Server-CAN concept ............................................ 109
6.3 Protocol basics ................................................ 111
6.4 Example basic run-time scenario .............................. 124
6.5 Advanced run-time mechanisms ............................... 126
6.6 Fault tolerance ................................................ 129
6.7 Summary ...................................................... 132

7 Server-CAN schedulers .............................................. 135
7.1 General Server-CAN properties ................................ 136
7.2 Periodic Server-Scheduled CAN (PS^2-CAN) .................. 139
7.3 Simple Server-Scheduled CAN (S^3-CAN) ..................... 144
7.4 Periodic Polling scheduler (PP-CAN) .......................... 164
7.5 Summary ...................................................... 165

8 Evaluation of Server-CAN schedulers ......................... 167
8.1 Simulation set up .............................................. 168
8.2 Simulation result format ...................................... 171
8.3 Evaluation of PP-CAN ......................................... 172
8.4 Evaluation of PS^2-CAN ...................................... 178
List of publications (short)

The full list of publications is presented in Appendix A. The following is the list of publications that are specifically used in the thesis, i.e., these publications forms the basis of the thesis (in reverse chronological order):


Swedish summary

Många av dagens produkter är sammansatta av ett antal mer eller mindre avancerade datorsystem. Till exempel består en bil av en mängd samverkande datorer som styr motorn, växellådan, bromsarna, bilstereo, mm. En trend inom bilindustrin och annan teknikintensiv industri är att gå från utveckling av flertalet datorsystem i isolering under i stort sätt egna förutsättningar och med egna resurser, till att datorsystemen utvecklas under gemensamma förutsättningar med delade resurser. Detta motiveras av ökad produktkomplexitet och krav på ökad produktionseffektivitet. Genom att integrera flera datorsystem kan man minska både kostnad och energiförbrukning för elektronik och kablage, samtidigt som produktkostnaden och utvecklingskostnaden kan hållas nere.

I denna avhandling presenteras metoder som underlättar integrering av datorsystem inbyggda i produkter. Metoderna begränsar hur de integrerade datorsystemen stör varandra, och däremot kommer de prestandagarantier som var givna innan integrering även gälla efter integrering. Metoderna som utvecklats löser detta problem för de nätverk som används för kommunikation mellan datorerna.

För att förklara den bakomliggande principen kan vi likna datornätverken vid flertilliga motorvägar. Vi kan t.ex. tänka oss en femfilig motorväg trafikerad en massa fordon av olika typ. För att undvika att fordon av en typ (t.ex. lastbilar) ska störa trafikrytmen för fordon av en annan typ (t.ex. personbilar) så delar den presenterade lösningen upp motorvägen med vajerräcken så att t.ex. två filer enbart kommer att kunna trafikeras av lastbilar och de övriga tre enbart av personbilar. En finess med lösningen – utöver att de olika fordonsstyperna inte kommer att kunna störa varandra – är att det inte spelar någon roll för t.ex. personbilstrafiken om det helt plötsligt börjar åka fordon av en tredje typ (t.ex. traktorer) i lastbilsfilerna. I termer av de kommunicerande datorerna innebär detta att ett delsystem utvecklat för en viss kommunikations-
kapacitet ( motsvarande ett visst antal motorvägsfiler) utan risk för störningar kan integreras med andra delsystem i olika produkter så länge den erforderliga kommunikationskapaciteten finns tillgänglig.
Chapter 1

Introduction

Integration of electronic subsystems (to form an integrated hardware and software system), is considered an important and complex issue in many application domains, such as, automotive, aerospace and consumer electronics [7, 103]. In addition, the time-to-market pressure, the emergence of multi-site development, the reuse of existing subsystems (legacy hardware and software), and the ever-increasing size of software, are driving radical changes in the development approaches of modern applications.

Both academic and industrial research and development, together with standardisation efforts, have over the years resulted in technologies, protocols, and engineering processes that facilitate a move from specialised implementations and solutions to more general and flexible solutions, supporting the evolution of modern embedded systems. However, one of the key problems in the embedded real-time systems domain is integration of subsystems. It is often the case that a modern embedded system comprises multiple (possibly) independently developed subsystems. In this thesis, predictable integration is defined as the complex issue of running two or more independently developed subsystems together on the same system architecture without having them interfere with each other, affecting their temporal performance, and potentially leading to violations of temporal requirements. In order to achieve a flexible yet predictable integration, efficient and flexible scheduling techniques can be used.

The technical contribution of this thesis is concerning scheduling of messages in embedded networks. Specifically, the concept of share-driven scheduling is evaluated and applied to the Controller Area Network (CAN) [77], re-
sulting in a share-driven scheduling framework called Server-CAN [152]. This scheduling framework is used to facilitate efficient subsystem integration, and can, for instance, be applied in the automotive domain to integrate Electronic Automotive Systems (EASs). The automotive domain is selected as a potential application domain for the Server-CAN framework due to its large number of subsystems. Moreover, CAN is one of the major embedded network technologies, used in multiple application domains, allowing for efficient event-triggered real-time communications.

1.1 Subsystem integration

Typically, a system comprises of many subsystems that can be independently developed, bought, and reused from earlier systems. When forming a new system, several subsystems are integrated together to form the system. This integration process can be more or less simple, depending on how the subsystems affect each other once they are integrated, and to what extent the subsystems have to be retested and verified.

An embedded network is a computer network found in embedded applications such as automotive, aerospace and consumer electronics. Messages are the carrier of data sent between a sender (user) and one or more receivers in these networks. Scheduling is the process of deciding which user is allowed to use a resource at what time. In general, several scheduling mechanisms can be used to schedule a resource. In this thesis, a user is defined as a stream of messages, and the resource is the embedded network that is shared among the users. The resource provided by the network is the capacity to send messages, i.e., the capacity of the communications channel. Hence, in this thesis, the scheduler decides which message that is to be transmitted (sent) on the network at what time. Typically, schedulers are classified as either priority-driven, time-driven or share-driven.

- A priority-driven scheduler schedules users based on their priority. The priority is a basis for providing timing guarantees, since it is used by the scheduler to decide the order in which messages are sent. Typically, a message that is frequently sent is assigned a high priority whereas a message sent more seldom is assigned a lower priority.

- Another way to schedule a resource is to divide time into partitions, called slots, and associate these slots with users. In this way a user will know which slot(s) it is allowed to use. A time-driven scheduler
1.2 Problem description

will make sure that users are only using the resource during their associated time slots. For example, a control application might consist of a number of periodic messages, i.e., messages with a fixed and known message inter-arrival time. Then, a static schedule can be created where the periodic messages are positioned in slots, fulfilling the timely message transmissions required by the control application.

• A third way to schedule a resource is by a share-driven scheduler. Here a share (fraction) of the resource is associated to a user. For example, one third of the resource can be allocated to the control system controlling the engine of a car. Hence, the messages involved in the engine control system are allowed to use a third of the bandwidth.

1.2 Problem description

A system consists of a number of nodes (computers) interconnected with a number of networks, forming a distributed system. On this distributed system, a number of subsystems are running together, sharing the system’s resources. When several subsystems are sharing the common distributed architecture of nodes and networks, they are affecting each others behaviour. This interference can in the worst case cause violations of the subsystems’ specified requirements, mandating a costly system level verification/testing even in cases when the subsystems are already thoroughly verified in isolation.

Looking at the nodes in the system, the subsystems may be affecting each others temporal behaviour and run-time environment. Concerning the temporal behaviour, this can be solved by providing temporal partitioning, i.e., separating the actual execution of the subsystems in time. Temporal partitioning is achieved by using time-driven or share-driven scheduling mechanisms, scheduling the usage of the resources (CPU and network) in time. As time-driven scheduling divide time into slots where a scheduled user may use the resource for the duration of these slots, users are temporally partitioned. Share-driven schedulers allow users to utilize a share of a resource. Hence, users sharing this portion of the resource are temporally partitioned from users sharing another portion of the resource. Concerning the run-time environment, the subsystems’ interference on each other can be handled by providing spatial partitioning. Spatial partitioning prevents interference between subsystems caused by, e.g., memory elements that are overwritten. However, the run-time environment is not the focus of this thesis.
From a network point of view, the integration of subsystems concerns only the scheduling of messages. Depending on the type of network in use (time-triggered, event-triggered network, or combined time- and event-triggered network) the scheduling of message transmissions is somewhat different.

- On a time-triggered network the messages are sent at specific instants in time, avoiding message transmission collisions, following a static schedule. This makes the message transmissions very deterministic and predictable, which is good from an analysability point of view. Integrating subsystems on a time-triggered network requires a static schedule to be created, fulfilling the subsystems’ requirements on message transmissions. The schedule provides temporal partitioning between the different subsystems. However, since a static schedule has to be built for each possible configuration of the system with its subsystems, time-triggered communication is somewhat inflexible when it comes to system evolution, e.g., changes and upgrades of the system.

- On event-triggered networks messages are sent at any given time (compared with the pre-defined slots of a time-triggered network). Message transmissions and (possible) collisions are resolved during runtime. Hence, an event-triggered network is scheduling messages according to a dynamic schedule created online, which makes message transmission-times somewhat harder to predict compared with time-triggered networks. However, this dynamic behaviour provides a greater degree of flexibility in the sense that there is no need to reconstruct the whole schedule each time a change has to be done, as all changes are taken care of by the run-time scheduler. Therefore, at a first glance integration of subsystems on an event-triggered network seems easier compared to on a time-triggered network. However, no temporal partitioning is provided, causing the subsystems’ message transmissions to interfere with each other.

- Mixed time- and event-triggered networks provide features of both paradigms. This is usually realised by dividing time into cycles and partitioning these cycles into time- and event-triggered windows where messages are sent accordingly. Also, hierarchical schedulers, where one scheduler is scheduling other schedulers, provide a mix of time- and event-triggered features. For example, an event-triggered scheduler can be used to schedule several time- and event-triggered schedulers that in turn schedule the messages.
1.3 Proposition and research questions

This thesis is about facilitating integration of subsystems from a network point of view. CAN, an event-triggered communication network popular in many application domains, is selected for the implementation of a share-driven scheduling framework called Server-CAN. As an example target application domain, the automotive domain is selected, both for its relevance for the research projects resulting in this thesis\textsuperscript{1,2} and for the wide variety of distributed embedded subsystems in this domain.

1.3 Proposition and research questions

Due to the nature of the research projects resulting in this thesis, the targeted application is found in the automotive domain. This leads to the following general question:

Q1 What characterise networked Electronic Automotive Systems (EASs)?

Q1 is a very general question that is broken down into the following four sub questions:

1 Which typical EASs exist?
2 What is the main issue with EAS based system development?
3 Which network technologies do these EASs use?
4 How are these network technologies scheduled?

Q1 and the problem description above boils down to a central proposition and a number of research questions:

P1 Share-driven scheduling of event-triggered networks facilitates efficient integration of electronic subsystems from a network point of view.

In the proposition, the property to achieve efficiency is that if subsystems are integrated with other subsystems, they will not interfere with each other. As a consequence, subsystems do not need to know the details of each other. This makes the integration process more efficient, as subsystems can be developed independently, and verifications done on the subsystem in isolation will by design hold also for the integrated system.

\textsuperscript{1}RATAD: http://www.mrtc.mdh.se/index.php?choice=projects&id=0038
\textsuperscript{2}SAVE: http://www.mrtc.mdh.se/save/
Based on proposition P1 and the answer to question Q1 three specific research questions are formulated:

**Q2 How can share-driven scheduling be implemented in the network technologies used by EASs?**

Firstly, answering Q1-3 and Q1-4, the network technologies used by EASs must be investigated to determine the requirements put by the most relevant network technologies. Following this, a number of design decisions have to be made in order to know where and how to implement a share-driven scheduler.

**Q3 What is the real-time performance of such an implementation?**

The real-time performance of the share-driven scheduler has to be analytically investigated and proven. Moreover, the temporal performance of the share-driven scheduler should be evaluated against existing schedulers. This can be done, for example, in a real system or in a simulation study.

**Q4 Does such an implementation facilitate integration of EASs?**

It has to be investigated how to use the share-driven scheduler in the context of integration. Typical automotive subsystem requirements are identified when answering Q1-1 and Q1-2. Once such subsystems are integrated using the share-driven scheduler their temporal performance has to be evaluated.

### 1.4 Research work and method

Answering the questions above, the research has been conducted by surveying the research community, attending relevant international symposia, conferences, workshops, and summer schools, surveying application domains, taking relevant courses, and talking with key persons both from industry and academia.

Based on this state-of-the-art, answers and solutions for the questions have been proposed, tested and evaluated. The technical soundness of the research is confirmed by mathematical proofs, simulation studies allowing for “what if” analysis of various properties, and discussions with both local and international
1.5 Contributions

colleagues. Quality has been assured by dissemination of research results in international peer-reviewed journals, conferences and symposia, workshops and work-in-progress sessions.

The research has been conducted both with colleagues in projects run at Mälardalen University, as well as together with international colleagues. Also, two longer international visits working with other research groups have been conducted. Additionally, parts of the material in this thesis was presented as a Licentiate thesis\(^3\) at Mälardalen University in 2003 [141].

1.5 Contributions

Answering Q1, the thesis begins with three background chapters that investigate Electronic Automotive Systems (EASs) and survey network technologies and corresponding network schedulers. These chapters are based on material presented in [60, 61, 142, 143, 144].

The main research contributions of this thesis are answering questions Q2, Q3 and Q4:

1. Answering Q2:

- **Development of a new server-based scheduling concept: the Server-CAN framework**

  The material presented in the introduction and background chapters is motivating the proposal of the Server-CAN framework [141, 148, 149, 152, 154]. Server-CAN is a higher layer share-driven scheduling concept for CAN that, compared to other CAN schedulers (time- and priority-driven), increase flexibility and facilitate subsystem integration by allowing share-driven scheduling of messages. Server-CAN is the first share-driven scheduler for CAN.

- **Development of two specialised Server-CAN schedulers: S\(^3\)-CAN and PS\(^2\)-CAN**

  The Server-CAN framework is used to implement two specialised Server-CAN schedulers, namely Periodic Server-Scheduled CAN (PS\(^2\)-CAN) and Simple Server-Scheduled CAN (S\(^3\)-CAN). PS\(^2\)-CAN is a simple Server-CAN scheduler, implementing a dynamic priority bandwidth conserving variant of the Polling Server (PS) [111, 190, 195], whereas S\(^3\)-CAN is a Server-CAN scheduler

---

\(^3\)A licentiate degree is a Swedish graduate degree halfway between M.Sc. and Ph.D.
implementing a bandwidth conserving polling version of the Total Bandwidth Server (TBS) [197, 199].

2. Answering Q3:
   - **Analysis of the real-time performance of S$^3$-CAN and PS$^2$-CAN**
     Targeting real-time systems, analysis of Server-CAN’s temporal performance [150, 151, 152] and worst-case real-time analysis are presented in detail and analytically proven for both PS$^2$-CAN and S$^3$-CAN.
   - **Evaluation of S$^3$-CAN and PS$^2$-CAN**
     In a simulation study, the temporal performance of PS$^2$-CAN and S$^3$-CAN is investigated in detail as well as evaluated in comparison with another CAN scheduler representing other existing higher layer CAN schedulers.

3. Answering Q4:
   - **Showing how the Server-CAN framework can be used in integration of electronic subsystems**
     It is shown how to use the Server-CAN framework in the context of subsystem integration. Specifically, it is shown how to configure the Server-CAN framework and how to analytically determine the feasibility of a subsystem once integrated.

1.6 **Thesis outline**

The thesis begins with four background chapters related to real-time systems, real-time communications, and Electronic Automotive Systems (EASs):

- **Chapter 2** introduces basic terms and concepts related to distributed real-time systems, real-time communications and real-time scheduling used and referred to throughout the thesis.
- **Chapter 3** investigates EASs in detail, presenting information on communications requirements, typical functions and subsystems used, as well as some real example architectures.
1.6 Thesis outline

- **Chapter 4** is surveying real-time communication networks in various application domains with a focus on the automotive domain. This chapter provides knowledge of which network technologies that are mostly used today in the automotive domain and which technologies that are expected to be used in the near future. CAN is identified as an important network technology for several domains, both today and for the future.

- CAN is presented in detail in **Chapter 5**, including history, standards, usage, schedulers and real-time performance. This chapter is motivated by the previous two chapters and puts the thesis contributions into the context of CAN.

Following the introduction and background chapters, the results of the research are presented in four chapters:

- **Chapter 6** introduces the Server-CAN framework, providing a general share-driven scheduler for CAN that facilitates subsystem integration of CAN-based systems from a network point of view. This chapter presents Server-CAN properties and features in detail, including basic and advanced run-time mechanisms.

- In **Chapter 7**, two Server-CAN schedulers named PS$^2$-CAN and S$^3$-CAN are presented in detail together with associated analytical worst-case response-time analysis. Also, a reference CAN scheduler is presented (the Periodic Polling scheduler (PP-CAN)).

- The reference scheduler is used for evaluation purposes in **Chapter 8**, where PP-CAN, PS$^2$-CAN and S$^3$-CAN are evaluated in an extensive simulation study.

- **Chapter 9** shows how to use the Server-CAN framework in the context of subsystem integration.

Finally, **Chapter 10** concludes the thesis, discusses results and future work.
Chapter 2

Theoretical background

The purpose of this chapter is to introduce the basic terms and concepts used throughout this thesis. In particular, real-time systems, real-time communications, real-time scheduling and real-time analysis are presented. The chapter is then summarised by putting these concepts and terms into the scope of this thesis.

2.1 Real-time systems

In real-time systems, not only the correct logical result of a computation is of importance, as real-time systems additionally depend on at which time the results of these computations are produced. Sometimes not only the time is of importance, as also the order in which computations are performed is significant. For example, two computations (e.g., A and B) could have what is called a precedence relation, requiring, for example, computation B to be finished before computation A. Over the last decades, various analysis techniques have been developed to determine at what time, in a worst-case scenario, a specific computation is completed.

Violating the predetermined properties in terms of timeliness can result in a bad or sometimes catastrophic scenario. For example, consider an airbag in a car. The purpose of an airbag system is to inflate a bag of gas that will catch the driver of a car in case of a collision. The airbag is not to be inflated too early nor too late, since this could cause injury to the driver. A too early inflation will allow too much gas to leave the airbag before the driver is thrown into it.
On the other hand, a too late inflation will cause the driver to smash into the steering wheel before the airbag is exploding in his/her face. Only the correct timing will cause the airbag to work as intended.

A real-time system can either be executing on a single CPU (single processor system) or it can be executing distributed over several CPUs. These CPUs can in turn be either tightly coupled with a shared memory (multiprocessor system) or loosely coupled with a distributed memory (distributed system), where the latter type of real-time system is the main focus of this thesis. These systems are often found in Distributed Computer Control Systems (DCCS), running in most application domains dealing with embedded real-time systems, e.g., automotive, avionic, trains, medical, and industrial automation. An overview of key issues in DCCS is presented in Figure 2.1 (figure inspired by Tovar [215]). This figure identifies nine components that contribute to the response-time of a task in network node A (task 1). The role of this task is to collect data from a process sensor located in the network node B. To do that, firstly task 1 has to be scheduled together with the other tasks on host processor A. Once task 1 is scheduled for execution, and it has executed until completion (1), its message is queued (2) in the communications adapter A before it is sent on the shared broadcast bus (3). When the message is received on node B, the communications adapter B is putting the message in the arrival-queue (4) before it can be consumed by task 3 (which must be scheduled (5) together with the other tasks on host processor B). Then, the procedure is repeated in the other direction (6-9) for the response.

### 2.1.1 Modelling real-time systems

In order to reason about a real-time system, a number of real-time system models have been developed that more or less accurately capture the temporal behaviour of the system.

A typical real-time system can be modelled as a set of real-time programs, each of which in turn consists of a set of tasks. These tasks are typically controlling a system in an environment of sensors, control functions and actuators, all with limited resources in terms of computation and communication capabilities. Resources such as memory and computation time are limited, imposing strict requirements on the tasks in the system (the system’s task set). The execution of a task is triggered by events generated by time (time events), other tasks (task events) or input sensors (input events). The execution delivers data to output actuators (output events) or to other tasks. Tasks have different properties and requirements in terms of time, e.g., Worst-Case Execution Times (WCET),
2.1 Real-time systems

periods, and deadlines. Several tasks can be executing on the same processor, i.e., sharing the same CPU. An important issue to determine is whether all tasks can execute as planned during peak-load. By enforcing some task model and calculating the total task utilisation in the system (e.g., the total utilisation of the CPU by all tasks in the system’s task set), or the response-time of all tasks in the worst-case scenarios (at peak-load), it can be determined if they will fulfil the requirement of completing their executions within their deadlines.

As tasks are executing on a CPU, when there are several tasks to choose from (ready for execution), it must be decided which task to execute. Tasks can have different priorities in order to, for example, let a more important task execute before a less important task. Moreover, a real-time system can be pre-emptive or non pre-emptive. In a pre-emptive system, tasks can pre-empt each other, allowing for the task with the highest priority to execute as soon as possible. However, in a non pre-emptive system a task that has been allowed to start will always execute until its completion, thus deferring execution of any higher priority tasks. The difference between pre-emptive and non pre-emptive execution in a priority scheduled system is shown in Figure 2.2. Here, two tasks, task A and task B, are executing on a CPU. Task A has higher priority than

![Figure 2.1: A Distributed Computer Control System (DCCS).](image)
task B. Task A is arriving at time 3 and task B is arriving at time 2. Scenarios for both non-pre-emptive execution (Figure 2.2a) and pre-emptive execution (Figure 2.2b) are shown. In Figure 2.2a the high priority task arrives at time 3 but is blocked by the lower priority task and can not start its execution until time 6 when the low priority task has finished its execution (blocking is explained further in Section 2.3.5). In Figure 2.2b the high priority task execute direct on its arrival at time 3, pre-empting the low priority task.

Before a task can start to execute it has to be triggered. Once a task is triggered it will be ready for execution in the system. Tasks are either event- or time-triggered, triggered by events that are either periodic, sporadic, or aperiodic in their nature. Due to this behaviour, tasks are modelled as either periodic, sporadic or aperiodic. The instant when a task is triggered and ready for execution in the system is called the task arrival. The time in between two arrivals of the same task (between two task arrivals) is called the task inter-arrival time. Periodic tasks are ready for execution periodically with a fixed inter-arrival time (called period). Aperiodic tasks have no specific inter-arrival time and may be triggered at any time, usually triggered by interrupts. Sporadic tasks, although having no period, have a known minimum inter-arrival time. The difference between periodic, sporadic, and aperiodic task arrivals is illustrated in Figure 2.3. In Figure 2.3 the periodic task has a period equal to 2, i.e., inter-arrival time is 2, the sporadic task has a minimum inter-arrival time of 1, and the aperiodic task has no known inter-arrival time.

The choice between implementing a particular part of the real-time system using periodic, sporadic or aperiodic tasks is typically based on the characteristics of the function. For functions dealing with measurements of the state of the controlled process (e.g., its temperature) a periodic task is typically used to
sample the state. For handling of events (e.g., an alarm) a sporadic task can be
used if the event is known to have a minimum inter-arrival time, e.g., an alarm
or the emergency shut down of a production robot. The minimum inter-arrival
time can be constrained by physical laws, or it can be enforced by some me-
chanical mechanism. If the minimum time between two consecutive events is
unknown, an aperiodic task is required for the handling of the event. While it
can be impossible to guarantee a performance of an individual aperiodic task,
the system can be constructed such that aperiodic tasks will not interfere with
the sporadic and periodic tasks executing on the same resource.

Moreover, real-time tasks can be classified as tasks with hard or soft real-
time requirements. The real-time requirements of an application spans a spec-
trum, as depicted in Figure 2.4 showing some example applications having
non-, soft-, and hard real-time requirements [202]. Hard real-time tasks have
high demands on their ability to meet their deadlines, and violation of these
requirements may have severe consequences. If the violation may be catas-
trophic, the task is classified as being safety-critical. However, many tasks
have real-time requirements although violation of these is not so severe, and in

![Figure 2.3: Periodic, sporadic, and aperiodic task arrival.](image)

![Figure 2.4: The real-time spectrum.](image)
some cases a number of deadline violations can be tolerated. Examples of real-time systems including such tasks are robust control systems and systems that contain audio/video streaming. Here, the real-time constraints must be met in order for the video and/or sound to appear good to the end user, and a violation of the temporal requirements will be perceived as a decrease in quality.

A central problem when dealing with real-time system models is to determine how long time a real-time task will execute, in the worst case. The task is usually assumed to have a Worst-Case Execution Time (WCET). The WCET is part of the real-time system models used when calculating worst-case response times of individual tasks in a system, or to determine if a system is schedulable using a utilisation based test. Determining the WCET is a research problem of its own, which has not yet been satisfactory solved. However, there exists several emerging techniques for estimation of the worst-case execution time [48, 168].

2.2 Real-time communications

Real-time communications aims at providing timely and deterministic communication of data between devices in a distributed system. In many cases, there are requirements on providing guarantees of the real-time properties of these transmissions. The communication is carried out over a communications network relying on either a wired or a wireless medium.

2.2.1 The ISO/OSI reference model

The objective of the ISO/OSI reference model [39, 228] is to manage the complexity of communication protocols. The model contains seven layers, depicted in Figure 2.5 together with one of its most common implementation, the TCP/IP protocol, and the model used in this thesis.

The lowest three layers are network dependent, were the physical layer is responsible for the transmission of raw data on the medium used. The data link layer is responsible for the transmission of data frames and to recognise and correct errors related to this. The network layer is responsible for the setup and maintenance of network wide connections. The upper three layers are application oriented, and the intermediate layer (the transport layer) isolates the upper three and the lower three layers from each other, i.e., all layers above the transport layer can transmit messages independent of the underlying network infrastructure.
2.2 Real-time communications

In this thesis the lower layers of the ISO/OSI reference model are of great importance, where for real-time communications, the Medium Access Control (MAC) protocol determines the degree of predictability of the network technology. Usually, the MAC protocol is considered a sub layer of the physical layer or the data link layer. In the following, a number of relevant (for real-time communications) MAC protocols are described.

2.2.2 Medium Access Control (MAC) protocols

A node with networking capabilities has a local communications adapter that mediates access to the medium used for message transmissions. Tasks that send messages send their messages to the local communications adapter. Then, the communications adapter takes care of the actual message transmission. Also, the communications adapter receives messages from the medium, delivering them to the corresponding receiving tasks (via the ISO/OSI protocol stack). When data is to be sent from the communications adapter to the wired or wireless medium, the message transmission is controlled by the medium access control protocols (MAC protocols).

Common MAC protocols used in real-time communication networks can be classified into random access protocols, fixed-assignment protocols and demand-assignment protocols. Examples of these MAC protocols are random access protocols such as

- CSMA/CD (Carrier Sense Multiple Access / Collision Detection),
Theoretical background

- CSMA/CR (Carrier Sense Multiple Access / Collision Resolution),
- CSMA/CA (Carrier Sense Multiple Access / Collision Avoidance),
and fixed-assignment protocols such as
- TDMA (Time Division Multiple Access),
- FTDMA (Flexible TDMA),
and demand-assignment protocols such as
- distributed solutions relying on tokens,
- centralised solutions by the usage of masters.

These MAC protocols are all used both for real-time and non real-time communications, and each of them have different timing characteristics. Below, all of these MAC protocols (together with the random access related MAC protocols ALOHA and CSMA) are presented together with a run-time example.

ALOHA

The classical random access MAC protocol is the ALOHA protocol [3]. Using ALOHA, messages arriving at the communications adapter are immediately transmitted on the medium, without prior checking the status of the medium, i.e., if it is idle or busy. Once the sender has completed its transmission of a message, it starts a timer and waits for the receiver to send an acknowledgement message, confirming the correct reception of the transmitted message at the receiver side. If the acknowledgement is received at the transmitter before the end of the timer, the timer is stopped and the message is considered successfully transmitted. If the timer expires, the transmitter selects a random backoff time and waits for this time until the message is retransmitted.

Consider the example in Figure 2.6, where message 1 arrives to its corresponding communications adapter at time 1 and directly initiates its transmission. However, at time 4 the arrival of message 2 at another communications adapter (that directly begins to transmit the message on the medium) destroys the transmission of message 1, causing both message 1 and message 2 to be retransmitted at some time in the future. In the example, message 2 is retransmitted at time 16 and message 1 is retransmitted at time 23. Note that the acknowledgement messages are not shown in the figure.
2.2 Real-time communications

ALOHA is a primitive random access MAC protocol with primary strength in its simplicity. However, due to the simplicity it is not very efficient and predictable, hence not suitable for real-time communications.

Carrier Sense Multiple Access (CSMA)

Improving the above mentioned approach of ALOHA is to check the status of the medium before transmitting [95], i.e., check if the medium is idle or busy before starting transmitting (this process is called carrier sensing). CSMA protocols do this and allow for ongoing message transmissions to be completed without disturbance of other message transmissions. If the medium is busy CSMA protocols wait for some time (the backoff time) before a transmission is tried again (different approaches exists, e.g., nonpersistent CSMA, p-persistent CSMA and 1-persistent CSMA). CSMA relies (as ALOHA) on the receiver to transmit an acknowledgement message to confirm the correct reception of the message.

However, the number of collisions is still high when using CSMA (although lower compared with ALOHA). Using pure CSMA the performance of ongoing transmissions is improved but still it is a delicate task to initiate a transmission when several communication adapters want to start transmitting at the same time. If several transmissions are started in parallel all transmitted messages are corrupted which is not detected until the lacking reception of a corresponding acknowledge message. Hence, time and bandwidth is lost.

Looking at the example in Figure 2.6, using CSMA the communications adapter transmitting message 2 would never initiate transmission if the medium is not idle. Instead, the communications adapter transmitting message 2 would try again at a later time, allowing message 1 to be transmitted until completion. However, if message 1 and message 2 would arrive at approximately the same time (to their corresponding communication adapters) there would be a
collision as they would be transmitted in parallel (since both communications adapters will find the medium idle).

An example of CSMA is shown in Figure 2.7, where both message 2 and message 3 are transmitted on the medium at the same time (as soon as the medium is idle after the transmission of message 1), causing a message collision. Finally, both message 2 and message 3 are retransmitted at time 18 and 23 respectively.

**Carrier Sense Multiple Access / Collision Detection (CSMA/CD)**

In CSMA/CD networks collisions between messages on the medium are detected by simultaneously writing the message and reading the transmitted signal on the medium. Thus, it is possible to verify if the transmitted signal is the same as the signal currently being transmitted. If they are not the same, one or more parallel transmissions are going on. Once a collision is detected the transmitting stations stop their transmissions and wait for some time (generated by the backoff algorithm) before retransmitting the message in order to reduce the risk of the same messages colliding again. However, due to the possibility of successive collisions, the temporal behaviour of CSMA/CD networks can be somewhat hard to predict. CSMA/CD is used, e.g., for Ethernet (see Section 2.2.3).

Looking at the example in Figure 2.7, the collision between message 2 and 3 will be detected immediately and their corresponding retransmissions can be initiated at an earlier time compared with when using CSMA without collision detection.
2.2 Real-time communications

Carrier Sense Multiple Access / Collision Resolution (CSMA/CR)

CSMA/CR does not go into a backoff mode (as the above mentioned approaches) once there is a collision detected. Instead, CSMA/CR resolves collisions by determining one of the message transmitters involved in the collision that is allowed to go on with an uninterrupted transmission of its message (see how this is done for CAN in Chapter 5). The other messages involved in the collision are retransmitted at another time, e.g., directly after the transmission of the first message. The same scenario using the CSMA/CD MAC protocol would cause all messages involved in the collision to be retransmitted.

Due to the collision resolution feature of CSMA/CR, it has the possibility to become more predictable in its temporal behaviour compared to CSMA/CD. An example of a network technology that implements CSMA/CR is CAN [77] (presented in Chapter 4 and Chapter 5).

Carrier Sense Multiple Access / Collision Avoidance (CSMA/CA)

In some cases it is not possible to detect collisions although it might still be desirable to try to avoid them. For example, using a wireless medium often makes it impossible to simultaneously read and write (send and receive) to the medium, as (at the communications adapter) the signal sent is so much stronger than (and therefore overwrites) the signal received. CSMA/CA protocols can avoid collisions by the usage of some handshake protocol in order to guarantee a free medium before the initiation of a message transmission. CSMA/CA is used by, e.g., ZigBee [72, 227] presented in Chapter 4.

The example depicted in Figure 2.8 shows a scenario with three messages, where message 1 arrives at its communications adapter at time 1 and initiates its transmission. At time 4, message 2 and message 3 arrives at two other communications adapters. However, the arrival of message 2 and message 3 does...
not cause all three messages to be retransmitted as with ALOHA (depicted in Figure 2.6), or retransmission of both message 2 and message 3 as would be the case using CSMA (depicted in Figure 2.7) or CSMA/CD. Instead, message 2 and message 3 will be transmitted once the medium is idle (free) after the transmission of message 1 at time 5, and after the transmission of message 2 at time 11. This is guaranteed by both CSMA/CR and CSMA/CA, using different approaches. Note that, in the example, the time needed by the handshake protocol of CSMA/CA is not shown in the figure, i.e., it is assumed to be zero.

**Time Division Multiple Access (TDMA)**

TDMA is a fixed assignment MAC protocol where time is used to achieve temporal partitioning of the medium. Messages are sent at predetermined instances in time, called message slots. Often, a schedule of slots is created offline (before the system is running), and this schedule is then followed and repeated online, but schedules can also be created online.

Due to the time slotted nature of TDMA networks, their temporal behaviour is very predictable and deterministic. TDMA networks are therefore very suitable for safety-critical systems with hard real-time guarantees. A drawback of TDMA networks is that they are somewhat inflexible, as a message can not be sent at an arbitrary time. A message can only be sent in one of the message’s predefined slots, which affect the responsiveness of the message transmissions. Also, if a message is shorter than its allocated slot, bandwidth is wasted since the unused portion of the slot cannot be used by another message. For example, suppose a message require only half of its slot (as message 1 in Figure 2.9), then 50% of the bandwidth in that slot is wasted, to be compared with a CSMA/CR network that is available for any message as soon as the transmission of the previous message is finished. One example of a TDMA real-time network is TTP/C [100, 217] (presented in Chapter 4), where offline
Flexible TDMA (FTDMA)

Another fixed assignment MAC protocol is the FTDMA. As regular TDMA networks, FTDMA networks avoid collisions by dividing time into slots. However, FTDMA networks are using a mini slotting concept in order to make more efficient use of the bandwidth, compared to a TDMA network. FTDMA is similar to TDMA with the difference in run-time slot size. In a FTDMA schedule the size of a slot is not fixed, but will vary depending on whether the slot is used or not. In case all slots are used in a FTDMA schedule, FTDMA operates the same way as TDMA. However, if a slot is not used within a small time offset $\Delta$ after its initiation, the schedule will progress to its next slot. Hence, unused slots will be shorter compared to a TDMA network where all slots have fixed size. However, used slots have the same size in both FTDMA and TDMA networks. Variants of mini slotting can be found in, e.g., Byteflight [19] and FlexRay [51] (presented in Chapter 4).

In the example depicted in Figure 2.10, two messages are ready for transmission. Message 1 is scheduled for slot 2 and message 2 is scheduled for slot 4. As the schedule is ran, no message transmission is initiated within $\Delta$ time units after the start of slot 1 at time 0, causing it to be terminated early initiating slot 2 at time 1. Message 1 is scheduled and sent in slot 2 followed by the initiation of slot 3 at time 3. As with slot 1, no message transmission is initiated causing slot 3 to terminate. At time 7 slot 4 is initiated and message 2 is transmitted. The rest of the slots following slot 4 are all terminated after $\Delta$. 

Figure 2.10: FTDMA message transmission.
due to no message transmission, causing less bandwidth to be lost compared with a TDMA solution.

**Tokens**

An alternative way of eliminating collisions on the network is to achieve mutual exclusion by the usage of token based demand assignment MAC protocols. Token based MAC protocols provide a fully distributed solution allowing for exclusive usage of the communications network to one transmitter (communications adapter) at a time.

In token networks only the owner of the (unique within the network) token is allowed to transmit messages on the network. Once the token holder is done transmitting messages, or has used its allotted time, the token is passed to another node. Examples of token protocols are e.g., the Timed Token Protocol (TTP) [130] or the IEEE 802.5 Token Ring Protocol [73]. Also, tokens are used by, e.g., PROFIBUS [68, 89, 166] (presented in Chapter 4).

Consider the example given in Figure 2.11, where the token (TK) is sent to one node at time 1. The token owner is then transmitting messages 1 and 2 at times 2 and 6 respectively, before the token is passed on to another node at time 12. This new token owner receives the token at time 13, and transmits messages 3 and 4 at times 13 and 16 respectively.

**Master/slave**

Another example of demand assignment MAC protocols is the centralised solutions relying on a specialised node called the master node. The other nodes in the system are called slave nodes. In master/slave networks, elimination of message collisions is achieved by letting the master node control the traffic on the network, deciding which messages are allowed to be sent and when.
2.2 Real-time communications

This approach is used in, e.g., LIN [115, 116], TTP/A [100, 218, 98] and PROFIBUS (presented in Chapter 4).

As an example, depicted in Figure 2.12, the master generates a schedule where messages 1 and 2 are scheduled for message transmission. Once the schedule is sent by the master and received by the slaves at time 1-2, messages 1 and 2 are transmitted by the slaves at times 2 and 6 respectively. Then, the master generates a new schedule which is again sent by the master and received by all slaves at time 12-13, and messages 3 and 4 are transmitted at times 13 and 16 respectively.

![Figure 2.12: A master/slave controlled message transmission.](image)

2.2.3 Networks

Communication network technologies considered in this thesis are either wired networks or wireless. The medium can be either wired, transmitting electrical or optical signals in cables or optical fibres, or wireless, transmitting radio signals or optical signals.

**Wired networks**

Wired networks, which are the more common type of networks in DCCSs, is in this thesis represented by two categories of networks: fieldbus networks and Ethernet networks.

**Fieldbus networks** Fieldbuses is a family of factory communication networks that has evolved during the 80s and 90s as a response to the demand to reduce cabling costs in factory automation systems [186]. By moving from a situation in which every controller has its own cables connecting its sensors to the controller (parallel interface), to a system with a set of controllers sharing a single network (serial interface), costs could be cut and flexibility could be
increased. Pushing for this evolution of technology was both the fact that the number of cables in the system increased as the number of sensors and actuators grew, together with controllers moving from being specialized with their own microchip, to sharing a microprocessor (CPU) with other controllers.

Fieldbuses were soon ready to handle the most demanding applications on the factory floor. Several fieldbus technologies, usually very specialized, were developed by different companies to meet the demands of their applications. Different fieldbuses are used in different application domains. Some application domains and their fieldbus technologies are presented in Chapter 4. Moreover, a comprehensive overview of the history and evolution of fieldbuses is given in [186].

**Ethernet networks** In parallel with the development of various (specialised) fieldbus technologies providing real-time communications for, e.g., avionics, trains, industrial and process automation, and building and home automation, Ethernet established itself as the de facto standard for non real-time communications. Comparing networking solutions for automation networks and office networks, fieldbuses were initially the choice for DCCSs and automation networks. At the same time, Ethernet evolved as the standard for office automation, and due to its popularity, prices on Ethernet based networking solutions dropped. A lower price on Ethernet controllers made it interesting to develop additions and modifications to Ethernet for real-time communications, allowing Ethernet to compete with established real-time networks.

Ethernet is not very suitable for real-time communications due to its handling of message collisions. DCCSs and automation networks require timing guarantees for individual messages. Several proposals to minimise or eliminate the occurrence of collisions on Ethernet have been proposed over the years. The stronger candidate today is the usage of a switched based infrastructure, where the switches separate collision domains to create a collision free network providing real-time message transmissions over Ethernet [65, 66, 87, 88, 194].

Other proposals providing real-time predictability using Ethernet include, e.g., making Ethernet more predictable using TDMA [101], offline scheduling [223] or Token algorithms [165, 220]. Note that a dedicated network is usually required when using tokens, where all nodes sharing the network must obey the token protocol (e.g., the Timed Token Protocol (TTP) [130] or the IEEE 802.5 Token Ring Protocol [73]). A different approach for predictability is to modify the collision resolution algorithm [107, 133, 173].

Other predictable approaches are, e.g., the usage of a master/slave concept as FTT-Ethernet [163] (part of the Flexible Time-Triggered (FTT) frame-
2.2 Real-time communications

work [6]), or the usage of the Virtual Time CSMA (VTCSMA) [46, 134, 224] protocol, where packets are delayed in a predictable way in order to eliminate the occurrence of collisions. Moreover, window protocols [225] are using a global window (synchronized time interval) that also remove collisions. The window protocol is more dynamic and somewhat more efficient in its behaviour compared to the VTCSMA approach.

Without modifications to the hardware or networking topology (infrastructure), the usage of traffic smoothing [30, 105, 106, 124] can eliminate bursts of traffic, which have severe impact on the timely delivery of message packets on the Ethernet. By keeping the network load below a given threshold, a probabilistic guarantee of message delivery can be provided.

For more information on real-time Ethernet interested readers are referred to [162].

Wireless networks

The wireless medium is often unpredictable compared to a wired medium in terms of the temporal behaviour of message transmissions. Therefore the temporal guarantees that can be provided with a wireless network are usually not as reliable as provided by a wired link. The reason for the lack of reliable timing guarantees is that the interference on the medium can not be predicted (and analytically taken into consideration) as accurately for a wireless medium as for a wired medium, especially interference from other sources than the communications network itself. Due to this unpredictability, no commercially available wireless communication networks are providing hard real-time guarantees. Still, wireless communication networks are being evaluated for usage in real-time applications. Examples of wireless networks that are used in the automotive domain are presented in Chapter 4.

2.2.4 Network topologies

There are different ways to connect the nodes in a wired distributed system. In this thesis, three different network topologies are considered, namely, bus, ring and star topology. Using a bus topology, all nodes in the distributed system are connected directly to the network. In a ring topology, each node in the distributed system is connected to exactly two other nodes in a specific way forming a ring of connected nodes. Finally, in a star topology all nodes in the distributed system are connected to a specific central node, forming a star. These three network topologies are depicted in Figure 2.13. Note that combi-
Different network topologies can exist, for example, a ring or a star might be connected to a bus together with other nodes.

### 2.3 Real-time scheduling

A real-time scheduler schedules real-time tasks sharing a resource (e.g., a CPU or a network link). The goal of the real-time scheduler is to ensure that the timing constraints of these tasks are satisfied. The scheduler decides, based on the task timing constraints, which task to execute or to use the resource at any given time.

Traditionally, real-time schedulers are divided into offline and online schedulers. Offline schedulers make all scheduling decisions before the system is executed. At run-time a simple dispatcher is used to activate tasks according to the schedule generated before run-time. Online schedulers, on the other hand, make scheduling decisions based on the system’s timing constraints during run-time.

As there are many different schedulers developed in the research community [28, 188], only the basic concepts of different types of schedulers are presented in this chapter.

In this thesis real-time schedulers are divided into three categories: time-driven schedulers, priority-driven schedulers and share-driven schedulers. This classification of real-time schedulers is depicted in Figure 2.14.

Note that there also exist combinations of the predictable time-driven schedulers and the more flexible priority-driven schedulers, and there exists methods to convert one policy to another [52, 129].
2.3 Real-time scheduling

Time-driven scheduling [99] work in the following way: The scheduler creates a schedule (sometimes called the table). Usually the schedule is created before the system is started (offline), but it can also be done during run-time (online). At run-time, a dispatcher follows the schedule, and makes sure that tasks are only executing at their predetermined time slots.

By creating a schedule offline, complex timing constraints, such as irregular task arrival patterns and precedence constraints, can be handled in a predictable manner that would be difficult to do online during run-time (tasks with precedence constraints require a special order of task executions, e.g., task A must execute before task B). The schedule that is created offline is the schedule that will be used at run-time. Therefore, the online behaviour of time-driven schedulers is very predictable. Because of this predictability, time-driven schedulers are the more commonly used schedulers in applications that have very high safety-critical demands, e.g., in avionics. However, since the schedule is created offline, the flexibility is very limited, in the sense that as soon as the system will change (due to, e.g., adding of functionality or change of hardware), a new schedule has to be created and given to the dispatcher. To create a new schedule can be non-trivial and sometimes very time consuming motivating the usage of priority-driven schedulers described below.

Figure 2.14: Real-time schedulers.
2.3.2 Priority-driven scheduling

Scheduling policies that make their scheduling decisions during run-time are classified as online schedulers. These schedulers make their scheduling decisions online based on the system’s timing constraints, such as, task priority.Schedulers that base their scheduling decisions on task priorities are called priority-driven schedulers.

Using priority-driven schedulers the flexibility is increased (compared to time-driven schedulers), since the schedule is created online based on the currently active tasks’ properties. Hence, priority-driven schedulers can cope with changes in work-load as well as adding and removing of tasks and functions, as long as the schedulability of the complete task-set is not violated. However, the exact behaviour of priority-driven schedulers is harder to predict. Therefore, these schedulers are not used as often in the most safety-critical applications.

Priority-driven scheduling policies can be divided into Fixed Priority Schedulers (FPS) and Dynamic Priority Schedulers (DPS). The difference between these scheduling policies is whether the priorities of the real-time tasks are fixed or if they can change during execution (i.e., dynamic priorities).

Fixed priority schedulers

When using FPS, once priorities are assigned to tasks they are not changed. Then, during execution, the task with the highest priority among all tasks that are available for execution is scheduled for execution. Priorities can be assigned in many ways, and depending on the system requirements some priority assignments are better than others. For instance, using a simple task model with strictly periodic non-interfering tasks with deadlines equal to the period of the task, a Rate Monotonic (RM) priority assignment has been shown by Liu and Layland [121] to be optimal in terms of schedulability. In RM, the priority is assigned based on the period of the task. The shorter the period is the higher priority will be assigned to the task.

Dynamic priority schedulers

The most well known DPS is the Earliest Deadline First (EDF) scheduling policy [121]. Using EDF, the task with the nearest (earliest) deadline among all tasks ready for execution gets the highest priority. Therefore the priority is not fixed, it changes dynamically over time. For simple task models, it has been shown that EDF is an optimal scheduler in terms of schedulability [43]. Also, EDF allows for higher schedulability compared with FPS. Schedulability is in
2.3 Real-time scheduling

the simple scenario guaranteed as long as the total load in the scheduled system is \( \leq 100\% \), whereas FPS in these simple cases has a schedulability bound of about 69\% (more on this in Section 2.4). For a good comparison between RM and EDF interested readers are referred to [27].

Other DPS are Least Laxity First (LLF) (sometimes also called Least Slack Time first (LST)) [114, 132]. Here the priorities of the tasks are generated at the time of scheduling from the amount of laxity (for LLF, or slack for LST) available before the deadline is violated. Laxity (or slack time) is defined as the maximum time a task can be delayed on its activation and still complete within its deadline [28].

2.3.3 Share-driven scheduling

Another way of scheduling a resource is to allocate a share [203] of the resource to a user or task. This is useful, for example, when dealing with aperiodic tasks when their behaviour is not completely known. Using share-driven scheduling it is possible to allocate a fraction of the resource to these aperiodic tasks, preventing them from interfering with other tasks that might be scheduled using time-driven or priority-driven scheduling techniques.

In order for the priority-driven schedulers to cope with aperiodic tasks, different service methods have been presented. The objective of these service methods is to give a good average response-time for aperiodic requests, while preserving the timing constraints of periodic and sporadic tasks. These services can be implemented as share-driven scheduling policies, either based on General Processor Sharing (GPS) [158, 159] algorithms, or using special server-based schedulers, e.g., [1, 2, 109, 111, 174, 175, 176, 190, 195, 197, 198, 204]. In the scheduling literature many types of servers are described, implementing server-based schedulers. In general, each server is characterised partly by its unique mechanism for assigning deadlines (for DPS based servers), and partly by a set of parameters used to configure the server. Examples of such parameters are priority (for FPS based servers), bandwidth, period, and capacity.

Share-driven scheduling in fixed priority systems

Several server-based schedulers for FPS systems exist where the simplest one is the Polling Server (PS) [111, 190, 195]. A polling server allocates a share of the CPU to its users. This share is defined by the server’s period and capacity, i.e., the PS is guaranteed to allow its users to execute within the server’s capacity during each server period. The server is scheduled according to RM together
with the normal tasks (if existing) in the system. However, a server never executes by itself. A server will only mediate the right to execute for its users, if some of its users have requested to use the server’s capacity. Otherwise the server’s capacity will be left unused for that server period. However, if the PS is activated and no user is ready to use the server capacity, the capacity is lost for that server period and the server’s users have to wait to the next server period to be served. Hence, the worst-case service a user can get is when it requests capacity right after the server is activated (with its capacity replenished). The behaviour of a PS server is in the worst-case equal to a task with the period of the server’s period, and a worst-case execution time equal to the server’s capacity. Hence, the analysis of a system running PS is straightforward.

Another server-based scheduler for FPS systems that is slightly better than the PS (in terms of temporal performance) is the Deferrable Server (DS) [111, 204]. Here, the server is also implemented as a periodic task scheduled according to RM together with the (if existing) other periodic tasks. The difference from PS is that the server is not polling its users, i.e., checking if there are any pending users each server period and if not drop all its capacity. Instead, the DS preserves its capacity throughout the server period allowing its users to use the capacity at any time during the server period. As with the PS, the DS replenish its capacity at the beginning of each server period. In general, the DS is giving better response times than the PS. However, by allowing the servers’ users to execute at any time during the servers’ period it violates the rules govern by the traditional RM scheduling (where the highest priority task has to execute as soon it is scheduled), lowering the schedulability bound for the periodic task set. A trade-off to the DS allowing a higher schedulability but a slight degradation in the response times is the Priority Exchange (PE) algorithm [111]. Here the servers’ capacities are preserved by exchanging it for the execution time of a lower priority periodic task. Hence, the servers’ capacities are not lost but preserved at the priority of the low priority task involved in the exchange. Note that the PE mechanisms are computationally more complex than the DS mechanisms, which should be taken into consideration in the trade-off.

By changing the way capacity is replenished, the Sporadic Server (SS) [195] is a server-based scheduler for FPS systems that allows high schedulability without degradation. Instead of replenishing capacity at the beginning of the server period, SS replenishes its capacity once the capacity has been consumed by its users. As DS, SS violates the traditional RM scheduling by not executing the highest priority task once it is scheduled for execution. However, this violation does not impact on the schedulability as the same schedulability bound is offered for a system running both with and without SS.
There are server-based schedulers for FPS systems having better performance in terms of response-time. However, this usually comes at a cost of high computational and implementation complexity as well as high memory requirements. One of these schedulers is the Slack Stealer [109, 174, 175, 176]. It should be noted that there are no optimal algorithms in terms of minimising the response time. The non existence of an algorithm that can both minimise the response time offered to users and at the same time guarantees the schedulability of the periodic tasks has been proven in [209]. Hence, there is a trade-off between response-time and schedulability when finding a suitable server-based scheduler for the intended target system.

**Share-driven scheduling in dynamic priority systems**

Looking at DPS systems, a number of server-based schedulers have been developed over the years. Many of the server-based schedulers for FPS systems have also been extended to EDF based DPS systems, e.g., an extension of PE called the Dynamic Priority Exchange (DPE) [197, 198], and an extension of the SS called the Dynamic Sporadic Server (DSS) [197, 198]. A very simple (implementation wise) server-based scheduler that provides faster response-times compared with SS yet not violating the overall load of the system (causing other tasks to miss their deadlines) is the Total Bandwidth Server (TBS) [197, 198]. TBS makes sure that the server never uses more bandwidth than allocated to it (under the assumption that the users do not consume more capacity than specified by their worst-case execution times), yet providing a fast response time to its users (i.e., assigning its users with a close deadline as the system is scheduled according to EDF). Also, TBS has been enhanced by improving its deadline assignment rule [28]. A quite complex server-based scheduler is the Earliest Deadline Late server (EDL) [197, 198] (which is a DPS version of the Slack Stealer). Moreover, there is an Improved Priority Exchange (IPE) [197, 198] which has similar performance as the EDL [28], yet being less complex implementation wise. When the worst-case execution times are unknown, the Constant Bandwidth Server (CBS) [1, 2] can be used, guaranteeing that the server’s users will never use more than the server’s capacity.

**2.3.4 Hierarchical schedulers**

In some cases parts of the system might better utilise one scheduling policy whereas other parts of the system would benefit more from using other schedul-
Theoretical background

ing policies. In fact, using server-based schedulers, the share provided by the server could be scheduled in a suitable way at the same time as the rest of the system is relying on FPS or DPS. In this way it is possible to construct hierarchies of schedulers, providing hierarchical scheduling to the system.

Examples of hierarchical schedulers for FPS systems are relying on Sporadic Servers (SS) as presented in [104], or relying on SS or Deferrable Servers (DS) as presented in [185], or relying on Polling Servers (PS) as presented in [119]. A more recent work [38] investigates hierarchical scheduling using all three SS, DS and PS, where an interesting result is the strong performance by the simplest server-based scheduler, the PS.

Looking at DPS, some hierarchical schedulers have been presented based on EDF, e.g., one two-level hierarchical scheduler [41, 42] which is hierarchically scheduling the Total Bandwidth Server (TBS) on an EDF scheduler. Another two-level scheduler [118] uses the Constant bandwidth Server (CBS), and a third is presented as the Bandwidth Sharing Server (BSS) [117] scheduling algorithm.

2.3.5 Scheduling with shared resources

When several tasks are sharing a resource, sometimes only one task is allowed to use the resource at a time. For example, a shared resource might be a data structure that the task is reading and writing to. If several tasks would be allowed to read and write to this data structure at the same time, its contents could become inconsistent. To ensure consistency of the shared resource it can be treated as a critical section which shall be protected by a mechanism that provides mutual exclusion. Mutual exclusion guarantees that only one task uses the resource at any time. Mutual exclusion may be implemented using, e.g., semaphores. A semaphore is a data structure (usually provided by the real-time kernel [200]) that can be accessed by primitives (also provided by the kernel) called wait and signal. When a task wants to use the shared resource, it executes a wait primitive on the semaphore causing the task to wait for the semaphore to be available (free). Several tasks waiting for the same semaphore are put in a queue. Whenever a task is done using the shared resource, it executes a signal primitive on the semaphore, making the semaphore available (free) again, allowing the next waiting task to take the semaphore. These two kernel primitives guarantees that only one task may have the semaphore at any time, hence providing mutual exclusion for the shared resource associated with the semaphore.

When mutual exclusion mechanisms are used, tasks are waiting for a shared
resource to be available before using it. Tasks waiting for a shared resource are blocked by the tasks currently using the resource. If a low priority task is in a critical section it owns the semaphore for that critical section. If a higher priority task wants to enter the same critical section as the one currently occupied by the low priority task, the high priority task has to wait (i.e., it becomes blocked) until the semaphore is released by the low priority task. This is called a priority inversion.

Depending on how a shared resource is used, in the worst case there might be a deadlock in the system. For example, suppose there are two shared resources, each protected by a semaphore, and two tasks. First task A gets semaphore 1 at the same time as task B gets semaphore 2. Then task A waits for semaphore 2 and task B waits for semaphore 1. In this scenario, both task A and task B are waiting for each other and nothing will resolve this state. Hence, the system is deadlocked. However, there are ways to avoid deadlocks and the easiest way is to enforce that all tasks in the system must wait for semaphores in the same order. For example, enforcing that both task A and task B would take their semaphores in the same order would avoid the deadlock scenario described above. However, there are more sophisticated ways of handling critical sections and in some cases also avoiding deadlocks.

**Priority Inheritance Protocol (PIP)**

One protocol that can be used in FPS systems is the Priority Inheritance Protocol (PIP) [191, 192]. Here, the priority of the task currently using the critical section might get a priority higher than its original priority. This will ensure that a higher priority task will not suffer from priority inversions caused by medium priority tasks. When a task would like to enter a critical section that is currently used by a lower priority task, the higher priority task is blocked by the lower priority one. When the higher priority task is blocked, and PIP is used, it transmits its active priority to the task that holds the semaphore. In this way a task may inherit a priority from a higher priority task. In general, the task currently in the critical section can cause several other tasks to be blocked, and will then inherit the highest priority of all the blocked tasks. Hence, whenever a task is blocking other tasks waiting for some critical section(s), it will inherit the highest priority among the tasks waiting for the critical section where the task is currently executing. Also, the priority inheritance is transitive, i.e., if task A is blocking task B, and task B is blocking task C, then task A will inherit the priority of task C.
Priority Ceiling Protocol (PCP)

Another protocol suitable for FPS systems is the Priority Ceiling Protocol (PCP) [170, 192], which prevents both priority inversion and deadlocks. Basically, PIP is extended such as a task is not always allowed to enter a critical section even if the semaphore is free. This is done by introducing a rule that guarantees that once a task has entered a critical section it can never be blocked by a lower priority task until it has executed until completion. Each semaphore is assigned a priority ceiling that is equal to the highest priority task that may lock the semaphore. Then a task is allowed to enter a critical section only if its priority is higher than that of all priority ceilings of all currently locked semaphores (by other tasks than itself) in the system.

Stack Resource Policy (SRP)

For DPS systems, some of the abovementioned methods have been extended, e.g., the Dynamic Priority Inheritance (DPI) [201] and the Dynamic Priority Ceiling (DPC) [33]. However, the most popular [188] is the Stack Resource Policy (SRP) [12].

When using SRP, a task may not pre-empt any other tasks until its priority is the highest among all tasks that are ready to run, and its pre-emption level is higher than the system ceiling. The pre-emption level of a task is a static parameter assigned to the task at its creation, and associated with all instances of that task. A task can only pre-empt another task if its pre-emption level is higher than the task that it is to pre-empt. Each resource in the system is associated with a resource ceiling and based on these resource ceilings, a system ceiling can be calculated. The system ceiling is a dynamic parameter that changes during system execution.

Share-driven scheduling with shared resources

Some of the more recent work on server-based schedulers are focused on extending CBS to also handle shared resources. Two of these extensions are the BandWidth Inheritance protocol (BWI) [120] and the Constant Bandwidth Server with Resource constraints (CBS-R) [29]. BWI allows for shared resources using a PIP mechanism. CBS-R is using the SRP in order to cope with shared resources. CBS-R is scheduling the whole system using servers, where each task has its own server.
2.4 Real-time analysis

Time-driven schedulers create a schedule offline. As the schedule is created, the schedule is verified so that all timing constraints are met. However, both priority-driven and share-driven schedulers have a more dynamic behaviour since the scheduling is performed during run-time. Here, timing analysis (schedulability tests) can be used in order to determine whether the temporal performance of a real-time system can be guaranteed for a certain task set scheduled by a certain scheduler. If such a guarantee is possible, the task set is said to be feasible.

There exist three different approaches for pre-run-time schedulability analysis: utilisation-based tests, demand-based tests and response-time tests. The first approach is based on the utilisation of the task-set under analysis (utilisation-based tests), the second is based on the processor demand at a given time interval (demand-based tests), and the third approach is based on calculating the worst-case response-time for each task in the task-set (response-time tests). Utilisation-based tests are usually less complex and faster to perform compared with demand-based tests and response-time tests, but they can not always be used for complicated task models [210, 212].

2.4.1 Task model

The task model notation used throughout this thesis is presented in Table 2.1.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( N )</td>
<td>Number of tasks in the task set</td>
</tr>
<tr>
<td>( C )</td>
<td>Worst-Case Execution Time (WCET)</td>
</tr>
<tr>
<td>( T )</td>
<td>Period</td>
</tr>
<tr>
<td>( r )</td>
<td>Release time</td>
</tr>
<tr>
<td>( D )</td>
<td>Relative deadline</td>
</tr>
<tr>
<td>( d )</td>
<td>Absolute deadline</td>
</tr>
<tr>
<td>( B )</td>
<td>Blocking-time</td>
</tr>
<tr>
<td>( R )</td>
<td>Response-time</td>
</tr>
<tr>
<td>( i )</td>
<td>Task under analysis</td>
</tr>
<tr>
<td>( hp(i) )</td>
<td>Set of tasks with priority higher than that of task ( i )</td>
</tr>
<tr>
<td>( lcp(i) )</td>
<td>Set of tasks with priority less than or equal to that of task ( i )</td>
</tr>
</tbody>
</table>

Table 2.1: Task model notation.
2.4.2 Definitions

**Definition 2.4.1.** Synchronous periodic tasks are a set of periodic tasks where all first instances are released at the same time, usually considered time zero [201].

**Definition 2.4.2.** Asynchronous periodic tasks are a set of periodic tasks where tasks can have their first instances released at different times [201].

**Definition 2.4.3.** Given by the critical instant theorem [121], a critical instant for a task $i$ is a release time $t$ for which the response time of task $i$ is maximised.

2.4.3 Utilisation-based tests

Seminal work on utilisation-based tests for both fixed priority schedulers and dynamic priority schedulers have been presented by Liu and Layland [121].

**Fixed priority schedulers**

In [121] by Liu and Layland, and earlier in [50, 187] by Fineberg and Serlin, a utilisation-based test for synchronous periodic tasks using the Rate Monotonic (RM) priority assignment is presented (Liu and Layland provided the formal proof/s). The task model they use consists of independent periodic tasks with deadline equal to their periods. Moreover, all tasks are released at the beginning of their period and have a known worst-case execution time and they are fully pre-emptive. If the test succeeds, the tasks will always meet their deadlines given that all the assumptions hold. The test is as follows:

$$\sum_{i=1}^{N} \frac{C_i}{T_i} \leq N \times \left(2^{1/N} - 1\right)$$

(2.1)

This test only guarantees that a task-set will not violate its deadlines if it passes this test. The lower bound given by this test is around 69% when $N$ approaches infinity. However, there are task-sets that may not pass the test, yet they will meet all their deadlines. Later on, Lehoczky showed that the average case “real” feasible utilization is about 88% when using random generated
task sets [110]. Moreover, Lehoczky also developed an exact analysis [108]. However, the test developed by Lehoczky is a much more complex inequality compared to Inequality 2.1. It has also been shown that, by having the task’s periods harmonic (or near harmonic), the schedulability bound is up to 100% [189]. Harmonic task sets have only task periods that are multiples if each other.

Inequality 2.1 has been extended in various ways, e.g., by Sha et al. [192] to also cover blocking-time, i.e., to cover for when higher priority tasks are blocked by lower priority tasks. For a good overview of FPS utilisation-based tests interested readers are referred to [188].

**Dynamic priority schedulers**

Liu and Layland [121] also present a utilisation-based test for EDF (with the same assumptions as for Inequality 2.1):

\[
\sum_{i=1}^{N} \frac{C_i}{T_i} \leq 1 \tag{2.2}
\]

This inequality is a necessary and sufficient condition for the task-set to be schedulable. Coffman [36] show that Inequality 2.2 is also valid for asynchronous task sets. However, later it has been shown that it is enough to investigate synchronous task sets in order to determine if a periodic task set is feasible or not [15].

**2.4.4 Demand-based tests**

The processor demand is a measure that indicates how much computation that is requested by the system’s task set, with respect to timing constraints, in an arbitrary time interval \( t \in [t_1, t_2] \). The processor demand \( h_{[t_1, t_2]} \) is given by

\[
h_{[t_1, t_2]} = \sum_{t_1 \leq r_k, d_k \leq t_2} C_k \tag{2.3}
\]

where \( r_k \) is the release time of task \( k \) and \( d_k \) is the absolute deadline of task \( k \), i.e., the processor demand is in an arbitrary time interval given by the tasks released within (and having absolute deadlines within) this time interval.
Looking at synchronous task sets, Equation 2.3 can be expressed as $h(t)$ given by

$$h(t) = \sum_{D_i \leq t} \left( 1 + \left\lceil \frac{t - D_i}{T_i} \right\rceil \times C_i \right)$$

(2.4)

where $D_i$ is the relative deadline of task $i$. Then, a task set is feasible iff

$$\forall t, h(t) \leq t$$

(2.5)

for which several approaches have been presented determining a valid (sufficient) $t$ [16, 15].

**Dynamic priority schedulers**

By looking at the processor demand, Baruah *et al.* [15] extend Inequality 2.2 to also allow for deadlines longer than the period. Moreover, Baruah *et al.* [16] present a processor demand-based feasibility test that allows for deadlines shorter than the period. Also, given that Inequality 2.2 is fulfilled, George *et al.* [58] introduce a generalized processor demand-based feasibility test that allows for non pre-emptive EDF scheduling. Additional extensions covering sporadic tasks is presented by Baruah *et al.* [15] and Zheng [226].

**2.4.5 Response-time tests**

Response-time tests are calculating the behaviour of the worst-case scenario that can happen for any given task, scheduled by a specific real time scheduler. This worst case behaviour is used in order to determine the worst-case response-time for that task.

**Fixed priority schedulers**

Joseph and Pandya presented the first response-time test for real-time systems [90]. They present a response-time test for pre-emptive fixed-priority systems. The worst-case response-time is calculated as follows:

$$R_i = I_i + C_i$$

(2.6)
where $I_i$ is the interference from higher priority tasks defined as:

$$I_i = \sum_{j \in hp(i)} \left( \left\lceil \frac{R_i}{T_j} \right\rceil \times C_j \right)$$

(2.7)

where $hp(i)$ is the set of tasks with higher priority than task $i$.

For FPS scheduled systems, the critical instant is given by releasing task $i$ with all other higher priority tasks at the same time, i.e., the critical instant is generated when using a synchronous task set [196]. Hence, the worst-case response-time for task $i$ is found when all tasks are released simultaneously at time 0.

The worst-case response-time is found when investigating the processors level-$i$ busy period, which is defined as the period preceding the completion of task $i$, i.e., the time in which task $i$ and all other higher priority tasks still not yet have executed until completion. Hence, the processors level-$i$ busy period is given by rewriting Equation 2.6 to:

$$R_i^{n+1} = \sum_{j \in hp(i)} \left( \left\lceil \frac{R_i^n}{T_j} \right\rceil \times C_j \right) + C_i$$

(2.8)

Note that Equation 2.8 is a recurrence relation, where the approximation to the ($n+1$)th value is found in terms of the $n$th approximation. The first approximation is set to $R_0^i = C_i$. A solution is reached when $R_i^{n+1} = R_i^n$, i.e., a so-called fixed-point iteration. The recurrence equation will terminate given that Inequality 2.2 is fulfilled [193, 196].

The work of Joseph and Pandya [90] has been extended by Audsley et al. [9] to cover for the non pre-emptive fixed-priority context. Note that non pre-emption introduces a blocking factor $B_i$ due to execution already initiated by lower priority tasks. As a lower priority task has started its execution, it can not be pre-empted; hence, it might block higher priority tasks for the duration of its worst-case execution time. Also, in a non pre-emptive system, the processors level-$i$ busy period is not including the task $i$ itself:

$$R_i^{n+1} = B_i + \sum_{j \in hp(i)} \left( \left\lceil \frac{R_i^n - C_i}{T_j} \right\rceil + 1 \right) \times C_j + C_i$$

(2.9)

where the blocking factor $B_i$ is defined as follows:
42 Theoretical background

\[ B_i = \begin{cases} 
0 & \text{if } \text{lep}(i) = \emptyset, \\
\max_{k \in \text{lep}(i)} \{ C_k - \varepsilon \} & \text{if } \text{lep}(i) \neq \emptyset.
\end{cases} \quad (2.10) \]

where \( \text{lep}(i) \) is the set of tasks with priority less or equal than task \( i \). \( \varepsilon \) is the minimum time quantum, which, in computer systems, corresponds to one clock cycle. Including \( \varepsilon \) makes the blocking expression less pessimistic by safely removing one clock cycle from the worst case execution time of task \( k \), causing the blocking [17]. The recurrence equation (Equation 2.9) is solved in the same way as Equation 2.8. Note that in the presence of blocking, the scenario giving the critical instant is re-defined. The maximum interference now occurs when task \( i \) and all other higher priority tasks are simultaneously released just after the release of the longest lower priority task (other than task \( i \)).

### Dynamic priority schedulers

In dynamic priority systems, the worst-case response-time for a task-set is not necessarily obtained considering the processors level-\( i \) busy period generated by releasing all tasks at time 0. Instead, Spuri [196] and George et al. [58] find the worst-case response-time in the processors deadline-\( i \) busy period. The deadline-\( i \) busy period is the period in which only tasks with absolute deadlines smaller than or equal to \( d_i \) are allowed to execute.

Hence, in dynamic-priority systems the worst-case response-time for an arbitrary task \( i \) can be found for the pre-emptive case when all tasks, but \( i \), are released at time 0. Then, multiple scenarios have to be examined where task \( i \) is released at some time \( t \).

Also for the non pre-emptive case, all tasks but \( i \) are released at time 0. However, one task with an absolute deadline greater than task \( i \) (i.e., one lower priority task) has initiated its execution at time \( 0 - \varepsilon \). Then, as in the pre-emptive case, multiple scenarios have to be examined where task \( i \) is released at some time \( t \).

Worst-case response-time equations for both preemptive and non-preemptive EDF scheduling are given by George et al. [58]. Furthermore, Palencia and González Harbour have extended the response-time analysis for EDF scheduled systems to include offsets [59, 156, 157].
2.5 Summary

This chapter gives an overview of the theoretical foundation for real-time systems with a focus on basic concepts and terms highly related to the core of the thesis, as the thesis deals with real-time systems with a focus on real-time communications. Typically, real-time communications deal with real-time scheduling of non pre-emptive message transmissions. Note that as the focus of this thesis is on communications, the general real-time issues concerning tasks discussed in this chapter are in the context of the rest of the thesis equivalents to messages.

In Chapter 3, the targeted application domain is presented accompanied by Chapter 4 and Chapter 5, which covers real-time communications found in this domain (the automotive domain), giving examples of all types of the networks, network topologies and communication techniques presented in this chapter. Chapter 5 presents one of the networking technologies in more detail, the Controller Area Network (CAN) [77]. CAN is then used in the later part of the thesis.

Following the initial chapters giving domain specific background information, the research contributions of the thesis are presented starting with Chapter 6. Here, Server-CAN [152] is presented, which is a share-driven real-time scheduler that deals with real-time scheduling of non pre-emptive message transmissions on a wired fieldbus type of real-time communications network. Here messages are scheduled by an online share-driven scheduler using the central masters’ concept on top of CSMA/CR, and a corresponding real-time analysis is presented. The Server-CAN scheduler supports periodic, sporadic and aperiodic message transmitters.
Chapter 3

Electronic automotive systems

An Electronic Automotive System (EAS) is a distributed computer system consisting of several subsystems. Each automotive subsystem consists of one or more Electronic Control Units (ECUs), and the number of ECUs in a modern automotive system can be up to 70, together distributing more than 2500 variables and signals [40, 112, 139]. The evolution of the number of networked ECUs are shown for four car manufacturers in Figure 3.1 (figure inspired by [112]). The large number of ECUs makes the automotive system complex in many ways, including how to support and manage the network interconnecting all ECUs. To manage this complexity and to support the automotive systems of tomorrow, the automotive industry has in recent years set up several large consortia in order to agree on a common scalable electric/electronic architecture (e.g., AUTOSAR\(^1\)) and a common scalable communication system (e.g., FlexRay\(^2\)). This will support and simplify future developments of EASs. As a next step many hydraulic automotive subsystems, such as steering and braking, will be replaced by communication networks (wires) and electric/electromagnetic sensors and actuators. These new solutions are commonly called x-by-wire systems, introducing more requirements on the network technologies.

The requirements of an automotive network technology originate from the

\(^1\)Automotive Open System Architecture (AUTOSAR). http://www.autosar.org/
applications and subsystems it has to support. These applications and subsystems are executing on a distributed embedded architecture (a good overview of automotive architectures and application requirements can be found in [11]). Major automotive subsystems that rely on communications are found in chassis, passive safety, powertrain, body and comfort electronics, multimedia and infotainment, x-by-wire, and wireless and telematics, all explained in more detail later in the chapter. Although not a subsystem in itself, diagnostics is heavily relying on networking. Today several different network technologies are used to address the various communication requirements set by these subsystems. To interconnect these systems there is a need for high bandwidth together with flexibility and predictability. Also, many subsystems are highly safety-critical, requiring predictable fault-tolerant operation.

An important issue that the automotive industry has to deal with is the high number of existing network technologies. From an engineering (and cost) perspective, it is desirable to use fewer and more general network technologies. To reduce the complexity it is desirable to commit to a limited set of network technologies that can be used in most of the applications. However, it is not likely (in the near future) that the number of network technologies can be reduced to only one, as such a technology would be forced to provide the properties supporting the most demanding automotive systems (in terms
of the communication requirements presented in Section 3.3). Having such a general network technology would probably make it too (unnecessary) expensive for simpler and less demanding systems. Hence, it is more likely that a few network technologies will be used, with different capabilities, allowing for a trade-off between performance and cost. Moreover, in order to support the automotive systems of tomorrow, these network technologies need to be interconnected, i.e., several different types of network technologies should be interconnected, providing timeliness, composability and fault tolerance across the whole “network of networks”.

### 3.1 Historical perspective

All automotive subsystems were initially connected by dedicated cables. Steering and braking were done using hydraulics and mechanics. However, as automotive systems became more complex, new lighter and smarter engineering solutions had to be found.

As the number of automotive subsystems relying on electronics increased, so were the cabling required for their interconnection. To reduce the amount of cabling, fieldbuses were introduced in the 70s and 80s. Using a fieldbus, several previously dedicated cables were replaced by a serial bus interconnecting the ECUs, decreasing both weight and cost of the automotive system. Today most ECUs communicate with each other using fieldbuses. The introduction of ECUs (i.e., embedded computers) in the automotive domain in the 80s has led to more advanced automotive subsystems, e.g., braking and vehicle dynamic subsystems that would be impossible to realise using only hydraulics and mechanics.

Sharing a fieldbus, the amount of cabling in automotive systems is drastically reduced. To incorporate fieldbuses in automotive subsystems, the automotive manufacturers initially developed their own fieldbus technologies. However, as many of the automotive manufacturers share subcontractors, there was a need for standardisations. One subcontractor, Bosch GmbH, developed one of the first network technologies that were standardised and intended for the automotive domain. This technology, the Controller Area Network (CAN) [77] (presented in detail in Chapter 5), was standardised in the beginning of the 90s and soon became the most used fieldbus in the automotive industry.
3.2 Current trends

The most recent technology advances push for replacing safety-critical hydraulic parts of automotive systems, such as steering and braking, with electronics. Thus, there is a need for new reliable high-speed fieldbus networks. Such fieldbuses have been developed and shown to work in several prototype cars, e.g., the Mercedes Benz F400\(^3\) or the Personal Mobility vehicle by Toyota\(^4\). These new “by-wire” solutions are commonly called x-by-wire systems.

There are several reasons for why the automotive producers are willing to replace hydraulics and mechanics with electronics. Maybe the most important reasons are cost, weight, and the technological limitation of such systems. Implementing new and more advanced functionality using hydraulics and mechanics will be too costly and complicated.

3.3 Communication requirements

In this section, a number of important general and technical requirements in the context of communications in electronic automotive systems are identified.

3.3.1 General requirements

Firstly, a major (general) requirement in the automotive domain is to keep the costs down [131]. Hence, all efforts to fulfil the requirements of an electronic automotive system must be balanced with the cost it takes to fulfil them to a certain sufficient level.

Secondly, the technologies used in the electronic automotive systems must be future proof. This means that the network technologies have to be available on the market for a long time as well as supporting the requirements set by the automotive subsystems of tomorrow. Examples of such requirements are, e.g., allowing for operation in harsh environments (automotive electromagnetic compatibility), support for power management (to conserve the vehicles battery power), support for both electrical and optical physical layers, and allowing for both low and high bandwidth [131].

Thirdly, the lead time (the time to market) is of great importance in a consumer oriented market as the automotive domain. Keeping the lead time down

\(^3\)http://www.mercedes-benz.com/
\(^4\)http://www.toyota.com/vehicles/future/pm.html
contributes to keeping the cost down (originating from development and capital) and increasing profits by an easier market introduction which generates better sales volumes [171].

Hence, the network technologies used in the automotive domain must be future proof and keep the costs down. However, these requirements very much depend on the subsystems using the automotive network and must be balanced.

### 3.3.2 Technical requirements

Today several different fieldbus technologies are used to address various technical communication requirements. In this thesis five requirements are highlighted and explained in more detail, namely fault tolerance, predictability, bandwidth, flexibility and security. Note that all these requirements must be balanced with the cost of fulfilling them to a certain acceptable level.

**Fault tolerance**

When the system does not behave according to its specification, the system’s incorrect behaviour is caused by faults. Fault tolerant communication systems are built so they are tolerant to consistent and inconsistent message faults (duplication and omission failures), defective circuits, line failures etc., and constructed using, e.g., redundant hard- and software architectures. Moreover, they should provide error containment, by using, for example, bus guardians to prevent the existence of babbling idiots [22, 205]. A babbling idiot is an unintended message transmitter, and a bus guardian prevents babbling idiots by restricting when a specific message can be sent. Generally speaking, a fault can cause an error which might result in a failure [10], e.g., the babbling idiot is a fault but the bus guardian prevents the error (the message) to cause a failure in the system.

**Predictability**

A predictable communication system provides guarantees in terms of timeliness, i.e., it makes it possible to know the transmission time of a message. Predictable communication requires correct delivery of messages. Many safety-critical automotive systems and subsystems also have strong real-time requirements which need predictability in order to be fulfilled. This can be solved by enforcing messages to be sent at predefined time instants (or within precise time intervals) to fulfil the intended subsystem functionality. Again, an
example is the airbag system, where the airbag has to be inflated at exactly the correct time in order to function properly, not too early nor too late.

**Bandwidth**

High bandwidth is required by many automotive subsystems. However, there is a trade-off between required bandwidth, the cost of providing such a bandwidth and the level of subsystem integration that is possible to achieve with a single shared communications network. In the automotive domain it is often more desirable to select a cheaper communications network with lower bandwidth due to strong requirements on cost. Also, low bandwidth lowers the risk of interference as the signalling on the medium is more robust. However, at a higher cost, the latest automotive network technologies provide high bandwidth allowing for the emerging automotive subsystems working together with high degree of system integration.

**Flexibility**

In order to handle the complexity of electronic automotive subsystems, the network technologies used must be flexible. Flexibility entails, for example, the ability to handle both event- and time-triggered messages, the possibility to cope with varying load and/or number of message flows on the network, as well as the potential for scalability and extensibility of the network. As presented in Chapter 2, in TDMA networks all message transmissions must usually be pre-determined offline, while in CSMA networks message transmissions are resolved online. The latter are often considered more flexible than the former. Some network technologies allow a combination of TDMA and CSMA message transmissions.

**Security**

When the communications is reachable from outside the automotive system (or by internal subsystems) by, e.g., diagnostics tools, wireless connections and telematics, it is important to ensure the security of the system, i.e., no unauthorized accesses to the system should be possible [97].
3.4 Typical subsystems

In an automotive system several subsystems rely on networking. In this thesis, these subsystems are classified into seven categories, namely chassis systems, passive safety systems, powertrain systems, body and comfort electronics, multimedia and infotainment, x-by-wire systems, and wireless and telematics. Below, these categories are described and examples of typical subsystems are given.

3.4.1 Chassis systems

Chassis systems are a part of the vehicle active safety systems, including driving dynamics and driver assistance functions such as

- **Antilock Braking System (ABS)** - which makes use of wheel speed sensors in order to prevent the wheels from locking in a brake situation. This helps the driver to maintain steering ability, avoiding skidding during braking.

- **Vehicle Dynamics Control (VDC)** - two examples of VDC are:
  - **Dynamic Stability Control (DSC)** - which improves vehicular driving dynamics relying on existing ABS sensors, a steering wheel sensor, some acceleration sensors, rotary movement sensors and brake pressure sensors.
  - **Electronic Stability Program (ESP)** - which is a further development of the ABS with additional sensors, such as steering wheel angle sensor and acceleration sensor. ESP is designed to assist the driver in skidding situations, such as over-steering, under-steering and roll-over situations [219].

- **Adaptive Cruise Control (ACC)** - which is a cruise control maintaining the vehicle speed set by its driver. However, compared to traditional cruise control (see body and comfort electronics below), ACC maintains a safe distance to the vehicle in front, by slowing down the vehicle if the vehicle in front is getting too close. Radar is used to measure the distance to the vehicle in front.

- **Electronic Damper Control (EDC)** - which adjusts the shock absorbers in real-time to changing road and driving conditions. This gives increased comfort, safety and driving performance of the vehicle. EDC
makes use of a steering wheel sensor, a speed sensor and some acceleration sensors.

All the above mentioned chassis systems require quite advanced control systems.

3.4.2 Passive safety systems

Passive safety systems, or vehicle passive safety systems, are controlling the operation of safety-related functions in the vehicle [21]. Examples of passive safety systems are

- **Airbags** - which are used to minimise injury to the driver and the passengers in a vehicle during a crash situation. Typically a vehicle contains several airbags (some models are known to have up to 12 airbags). These airbags are connected to sensors that detect abnormal situations, e.g., sudden vehicle acceleration or de-acceleration. Once an abnormal situation is detected, depending on the type of crash, the appropriate airbags are inflated in about half a millisecond after the crash detection.

- **Seat belt pretensioners** - which are used to pick up the slack and stretch the seat belt.

Both airbags and seat belt pretensioners are pyrotechnic devices triggered by the crash sensors.

3.4.3 Powertrain

Powertrain is the assembly by which power is transmitted from the engine of the vehicle, through the gear-box, to the driving axis. Powertrain functions include

- **Engine control** - which involves the coordination of fuel injection, engine speed, valve control, cam timing etc.

- **Electronic gear control** - which provides electronic coordination of the gears instead of a pure mechanical solution.
3.4 Typical subsystems

3.4.4 Body and comfort electronics

Body and comfort electronics comprises the largest number of ECUs in the vehicle. Some examples of body and comfort electronic functions are

- **Air condition and climate control** - which are controlling the in-vehicle climate, using temperature sensors, humidity sensors and feedback control.

- **Cruise Control (CC)** - which controls the speed of the vehicle, keeping the actual speed at a preset speed, maintaining this using feedback control.

- **Locks** - which controls the operation of locks in the vehicle, including all key locks, discrete control buttons within the vehicle, and the electromechanical locks in the doors.

- **Window lifts** - which controls the operation of windows in the vehicle, relying on motors and discrete control buttons usually located in the doors of the vehicle.

- **Seat control** - which controls the configuration of the vehicle driver seats, using discrete control buttons and motors.

- **Park distance control** - which assists the driver of the vehicle in a parking situation by using ultrasonic sonar distance sensors, indicating the distance to the nearest obstacle.

These systems typically rely on driver interaction and are not safety-critical, requiring discrete control and/or feedback control. They involve hundreds of system states and events, and interface to physical components in the vehicle, e.g., motors and switches (discrete control buttons).

3.4.5 Multimedia and infotainment

Multimedia and infotainment is an area of subsystems rapidly increasing in terms of software size, naturally since multimedia applications and contents consume lots of memory. Examples of these systems are (some without further explanation as they are well known) car stereos, audio systems and speakers, DVD players, GPS and navigation systems, monitors and displays, video games, and internet connectivity. Voice processing and speech recognition is used to provide a safe and convenient way of interaction with the vehicle, controlling multimedia and infotainment functions.
3.4.6 X-by-wire

X-by-wire is the notation for all new subsystems replacing hydraulic and mechanical parts with electronics and computer (feedback) control systems. In this chapter, x-by-wire systems are classified into non safety-critical x-by-wire systems and safety-critical x-by-wire systems. Non safety-critical x-by-wire systems include

- **Throttle-by-wire** - which is replacing the mechanical throttle linkage between the accelerator pedal and the engine with an electrical motor.

- **Shift-by-wire** - which is an electromechanical solution replacing the mechanical link between the automatic transmission and the shift lever.

These systems are non safety-critical as in case of losing the communication link, the throttle-by-wire can simply let the engine idle, and the shift-by-wire system can simply change the gearbox to neutral.

Historically, the throttle-by-wire system was the first among all x-by-wire systems, implemented in a Chevrolet Corvette series in 1980 [221]. Today, throttle-by-wire is present in most vehicles. Also, shift-by-wire (gear-by-wire) systems are implemented in many high-end vehicles [221].

However, for safety-critical x-by-wire systems a loss of communications could potentially lead to a disaster with resulting loss of life. Examples of safety-critical x-by-wire systems are

- **Brake-by-wire** - which is replacing the mechanical and hydraulic connections between the brake pedal at the brake actuators at the wheels with wires and electromechanical actuators.

- **Steer-by-wire** - which is an electromechanical solution replacing the direct link between the steering wheel and the front-wheel directional actuator. Steer-by-wire relies on steering wheel sensors, feed-back controllers to ensure correct steering angle force feedback to give the driver a familiar feeling, a motor based steering actuator, and pinion angle sensors.

Looking at the brake-by-wire and steer-by-wire systems found in vehicles today, they always have a mechanical backup. This since the concern of safety is of most importance, and the trust of the communications network is not sufficient. Moreover, the demand for this type of systems (by the customers) is not very high today as it is hard to realise the technical benefits of having an
3.4 Typical subsystems

x-by-wire system, even though the performance possible beats a mechanical solution. Both break-by-wire and steer-by-wire are waiting for the evolution of the electrical power of the vehicle, expected to increase from 14V to 42V [54, 91]. Optimistic forecasts say that this technology is not mature enough before 2010, and therefore safety-critical x-by-wire is on hold [221].

3.4.7 Wireless and telematics

Wireless and telematics is in this chapter a category representing subsystems used for interconnection of wireless devices as well as deployment of new functions relying on wireless connectivity.

Relying on wireless communications are

- **Laptop computers** - which are wirelessly connected with the vehicle, running powerful applications.
- **Cell phones** - which are connected wirelessly with the vehicle to provide continuous connection with the telecom infrastructure, e.g., the Internet.
- **GPS units** - which are providing vehicle position, direction and speed information anytime anywhere (but tunnels).
- **Car access systems** - which prevents unauthorised access to the vehicle.

Automotive telematic functions enable the deployment of a number of new services and applications integrating wireless network technology into a vehicle. As a result, the vehicle acquires new capabilities and offers more services to its users. Examples of telematic services and applications are

- **Navigation and traffic information systems** - which provide the driver of a vehicle equipped with a telematics unit with directions to a desired location, together with real-time traffic information for the route.
- **Advanced driver assistance** - which provide real-time traffic information by creation of ad-hoc networks of vehicle to vehicle (inter-vehicle) decentralised floating car data communication. These systems are intended to increase road safety, improving traffic flow, preventing traffic jams and avoiding accidents. It relies on communications between the vehicle and its surroundings, e.g., other vehicles and roadside objects.
- **Fleet management systems** - which allows for real-time tracking of vehicle location, speed and use. This is useful in logistics businesses.
56 Electronic automotive systems

- **Safety systems** - which include collision avoidance systems, unsafe driving profiling, and intelligent airbag deployment systems. Also, an automatic emergency call can be performed in the event of an accident, providing the emergency centre with information such as the location of the accident.

- **Security systems** - which provide vehicle antitheft and stolen vehicle tracking services.

- **Diagnostics and maintenance services** - which provide remote diagnostics and maintenance functions. These functions are relying on vehicle and driver monitoring. Diagnostics and maintenance services provide vehicle data vital for maintenance. Possible functions include real-time notification of when a vehicle is required to go to a service shop, and what should be done there as well as real-time inventory of spare parts.

- **Voice recognition and wireless Internet connection** - which allow drivers and passengers to receive and send voice-activated e-mails while on the road.

These are all functions relying on wireless network technologies for in-vehicle and inter-vehicle (vehicle-to-vehicle) communications. Also, for telematic functions, connection with the surrounding telecom communication backbone is needed, and in some cases also a GPS.

### 3.4.8 Diagnostics

Apart from the subsystems listed above, relying on networking, diagnostics is required by many vehicle functions such as emissions monitoring (enforced by law in some countries, e.g., On Board Diagnostics (OBD) [216]), diagnosing of components and properties, service and maintenance with the possibility of downloading and updating software.

### 3.4.9 Subsystems and requirements

Table 3.1\(^5\) shows how the above listed automotive subsystem categories are mapped with the communication requirements presented in Section 3.3.

---

\(^5\)Updated from [143] based on more automotive systems.
3.5 Example of automotive systems

To give some examples of contemporary automotive systems, three automotive system architectures are presented, namely Volvo XC90, BMW 7 series and VW Passat.

3.5.1 Volvo XC90

The automotive system architecture of the Volvo XC90 is presented in Figure 3.26. In the figure the network infrastructure of the XC90 is presented, while a selection of the corresponding ECU explanations is given in Table 3.2. All “blocks” in the figure represent one ECU.

The ECUs are divided into three groups: 1) powertrain and chassis, 2) body electronics and 3) infotainment. A total of around 40 ECUs are found in the XC90, and the Controller Area Network (CAN) is the most common network used to interconnect these ECUs. Also, the Local Interconnect Network (LIN) [115, 116] is used to connect slave nodes in the system. There are two CAN busses interconnected with each other using a gateway. The gateway is the Central Electronic Module (CEM) in the figure. The two CAN busses have different speeds. One 500Kbps “high speed” CAN network is used for powertrain and chassis, while a 125Kbps “low speed” CAN network is used for body electronics. MOST7 is used for infotainment.

Table 3.1: Automotive subsystems and their major requirements.

<table>
<thead>
<tr>
<th>Subsystem</th>
<th>Communication requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Fault tolerance</td>
</tr>
<tr>
<td>Chassis</td>
<td>YES</td>
</tr>
<tr>
<td>Passive safety</td>
<td>YES</td>
</tr>
<tr>
<td>Powertrain</td>
<td>YES</td>
</tr>
<tr>
<td>Body and comfort</td>
<td>SOME</td>
</tr>
<tr>
<td>Multimedia /</td>
<td></td>
</tr>
<tr>
<td>infotainment</td>
<td>NO</td>
</tr>
<tr>
<td>X-by-wire</td>
<td>YES</td>
</tr>
<tr>
<td>Wireless /</td>
<td>NO</td>
</tr>
<tr>
<td>telematics</td>
<td></td>
</tr>
<tr>
<td>Diagnostics</td>
<td>NO</td>
</tr>
</tbody>
</table>

6Courtesy of Volvo Car Corporation.
7MOST Cooperation: MOST - Media Oriented Systems Transport.
http://www.mostcooperation.com/
3.5.2 BMW 7 series

The automotive system architecture of the BMW 7 series can be found in [56]. From this material, the network technologies and their usage is presented in Table 3.3. Here, different classes of subsystems are presented together with typical properties and requirements found in these classes of subsystems. Note that K-CAN, F-CAN, PT-CAN and LoCAN are different CAN networks. Also, note that SI-BUS is a Byteflight [19] network.

The network infrastructure of the BMW 7 series is presented in Figure 3.3. The figure is based upon material presented in [56], although the different ECU functions are not explained here. Interested readers are referred to [56]. In the figure, CAN (“K-CAN”, “F-CAN”, “PT-CAN” and “LoCAN”) is used for chassis, powertrain and body and comfort systems. Byteflight (“SI-BUS”) is used for passive safety systems, and MOST is used for multimedia/infotainment.
3.5 Example of automotive systems

<table>
<thead>
<tr>
<th>Block</th>
<th>Powertrain and chassis</th>
<th>Block</th>
<th>Body electronics</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCM</td>
<td>Transmission control module</td>
<td>CEM</td>
<td>Central electronic module</td>
</tr>
<tr>
<td>ECM</td>
<td>Engine control module</td>
<td>SWM</td>
<td>Steering wheel module</td>
</tr>
<tr>
<td>BCM</td>
<td>Brake control module</td>
<td>DDM</td>
<td>Driver door module</td>
</tr>
<tr>
<td>BSC</td>
<td>Body sensor cluster</td>
<td>REM</td>
<td>Rear electronic module</td>
</tr>
<tr>
<td>SAS</td>
<td>Steering angle sensor</td>
<td>PDM</td>
<td>Passenger door module</td>
</tr>
<tr>
<td>SUM</td>
<td>Suspension module</td>
<td>CCM</td>
<td>Climate control module</td>
</tr>
<tr>
<td>DEM</td>
<td>Differential electronic module</td>
<td>ICM</td>
<td>Infotainment control module</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Block</th>
<th>Infotainment</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACD</td>
<td>Audio module</td>
</tr>
<tr>
<td>B1</td>
<td>Media player 1</td>
</tr>
<tr>
<td>B2</td>
<td>Media player 2</td>
</tr>
<tr>
<td>PHM</td>
<td>Phone module</td>
</tr>
<tr>
<td>MMM</td>
<td>Multimedia module</td>
</tr>
<tr>
<td>SUB</td>
<td>Subwoofer</td>
</tr>
<tr>
<td>ATM</td>
<td>Antenna tuner module</td>
</tr>
<tr>
<td>ICM</td>
<td>Infotainment control module</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Block</th>
<th>Multimedia/ infotainment</th>
<th>Body and comfort</th>
<th>Chassis</th>
<th>Powertrain</th>
<th>Passive safety</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Program size</td>
<td>100 MB</td>
<td>2.5 MB</td>
<td>4.5 MB</td>
<td>2 MB</td>
</tr>
<tr>
<td></td>
<td>ECUs</td>
<td>4 - 12</td>
<td>14 - 30</td>
<td>6 - 10</td>
<td>3 - 6</td>
</tr>
<tr>
<td></td>
<td>Bus</td>
<td>MOST</td>
<td>K-CAN</td>
<td>F-CAN/PT-CAN</td>
<td>LoCAN/PT-CAN</td>
</tr>
<tr>
<td></td>
<td>Bandwidth</td>
<td>22 Mbps</td>
<td>100 Kbps</td>
<td>500 Kbps</td>
<td>500 Kbps</td>
</tr>
<tr>
<td></td>
<td>Messages</td>
<td>660</td>
<td>300</td>
<td>180</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>Cycle time</td>
<td>20ms - 5s</td>
<td>50ms - 2s</td>
<td>10ms - 1s</td>
<td>10ms - 10s</td>
</tr>
<tr>
<td></td>
<td>Safety requirements</td>
<td>low</td>
<td>high/low</td>
<td>high</td>
<td>high</td>
</tr>
<tr>
<td></td>
<td>Bus topology</td>
<td>ring</td>
<td>bus</td>
<td>bus</td>
<td>bus</td>
</tr>
<tr>
<td></td>
<td>Transmission medium</td>
<td>fibre</td>
<td>cable</td>
<td>cable</td>
<td>cable</td>
</tr>
</tbody>
</table>

Table 3.2: ECU explanations of Volvo XC90.

Table 3.3: In numbers - Networking technologies of BMW 7 series.

3.5.3 VW Passat

The network infrastructure of a VW Passat is presented in Figure 3.4. The figure is based upon material presented in [113], although the different ECU functions are not explained here. Interested readers are referred to [113].

Here, as in the other two examples, different network technologies are used
for different classes of subsystems. Only CAN and LIN networks are used, although the differences among CAN networks are, for example, their different speeds (100 Kbps and 500 Kbps). For chassis and powertrain systems, “CAN Antrieb” is used, and “CAN Komfort” is used for body and comfort systems. For multimedia/infotainment, “CAN Infotainment” is used.

### 3.5.4 Subsystem integration

The XC90 contains around 40 ECUs, the BMW 7 series has around 65, and the VW Passat has around 45. Also, other car models are known to have up to 70 ECUs. Integrating subsystems on the distributed automotive architecture is becoming more and more complicated. To overcome this, as one example in the case of the XC90, Volvo is using the Volcano concept\(^8\) [11, 31, 171, 172]. The Volcano system provides tools for packaging data (signals) into network frames, both for CAN and LIN networks (more recent versions of Volcano also supports MOST and FlexRay [51] networks). This simplifies the design, development and maintenance of the automotive system. Using the Volcano tools it is also possible to perform a timing analysis of the system, needed at

---

\(^8\)The Volcano concept is developed by Volcano Communications Technologies AB, which in turn was Acquired by Mentor Graphics in May 2005.
3.6 Summary

Automotive systems are considered as a potential target application for the contributions presented in this thesis. A modern automotive system contains many subsystems such as advanced safety systems, powertrain control, sensors and means for diagnostics. These subsystems have evolved over time, relying on various communication services provided by different network technologies.

An interesting, and for this thesis relevant, property of the automotive domain is the large number of subcontractors used for developing the various subsystems composing a vehicle. This emphasises the need for capable methods and technologies supporting efficient integration of subsystems. However, a complicating factor is the large number of network technologies used in the automotive industry, although the number is starting to decrease to a smaller...
set of standards that are used also over the continents.

This chapter gives a short historical perspective on automotive communications, followed by communication requirements and a classification of automotive subsystems found in a modern car. Following this, a mapping between requirements and the classification of automotive subsystems is presented, and the chapter is ended by some example automotive systems.

It is shown how an automotive system consists of numerous subsystems of various types and requirements. Integrating these subsystems require efficient methods and supporting technologies. As CAN is the most common network technology in the automotive domain (as well as other application domains), the features of CAN impacts on this integration process. Hence, supporting mechanisms and technologies for integration of subsystems on CAN are needed. In Chapter 6, CAN is extended with a scheduler, the Server-CAN scheduling framework, allowing for efficient integration of subsystems on CAN. The integration of subsystems, using the Server-CAN scheduling framework, is presented in Chapter 9.
Chapter 4

Embedded networks

This chapter gives an overview of network technologies used in some of the major application domains targeted by this thesis. A focus is given to the automotive domain, with a thorough survey of network technologies. The automotive application domain is characterised by large volumes and high pressures on low cost per unit. Following the automotive domain, at the end of the chapter, a brief overview of network technologies found in avionics, trains, industrial and process applications, and in building and home automation systems are presented.

4.1 Automotive domain

The purpose of this section is to provide an overview of the different automotive network technologies used in the automotive domain today, identifying their key applications, their strengths and possible weaknesses. The network technologies are classified in three categories based on where their major applications are found: (1) current wired, (2) upcoming wired and (3) wireless. Within these categories a few technologies stand out as strong (future proof) candidates for the automotive systems of tomorrow. Interconnecting these network technologies also requires an efficient middleware. However, addressing this aspect in detail goes beyond the scope of this chapter. Interested readers are referred to [139].
4.1.1 Current wired technologies

Today several fieldbus technologies are used by different automotive system vendors. The three most common fieldbus technologies, namely LIN, CAN and Byteflight, are presented in detail. Then, the major network technology used in multimedia applications, MOST, is presented. Following this, a number of other related network technologies are briefly mentioned.

LIN

The Local Interconnect Network (LIN), was initiated in October 1998 by a consortium of automotive companies (Audi, BMW, Daimler-Chrysler, Volvo Communications Technologies, Volvo and Volkswagen) together with Motorola, and a first specification draft was released in July 1999. Following this, LIN was standardised (open standard) in 2000 (LIN 1.1), 2002 (LIN 1.3 [115]) and 2003 (LIN 2.0 [116]), and in 2001 LIN was introduced in its first production car series. Today LIN holds a strong position in the automotive application domain where it coexists well with CAN.

LIN is typically used in body and comfort electronics to control devices such as seat control, light sensors and climate control. For example, one subsystem could be a door of a car with all its functionality such as window lifts, door locks etc. These subsystems are then interconnected with the car’s main CAN network via a LIN/CAN gateway. LIN is often used together with CAN, as LIN complements CAN by being much cheaper and simpler yet supporting the communications needed for typical non safety-related automotive subsystems.

LIN is a master/slave time-triggered type of fieldbus with one master node and several slave nodes. The LIN master node is typically also acting as a LIN/CAN gateway allowing for the interconnection of the LIN network with other networks. The master node keeps a schedule for all message transmissions on the network. The schedule is generated before start of the system (offline), and it is managed during run-time by the master node. Basically, the master is polling the other nodes by sending a message header that contains a message identifier. The scheduled slave node will recognise this header and send its corresponding data. LIN provides 64 different message identifiers (6 bit message ID), where the ID denotes message contents, not physical address.

LIN is an inexpensive network providing network speeds of up to 20 Kbps.

---

where messages are sent in frames containing 2, 4 or 8 bytes. LIN is running on any UART/SCI, using a single wire, and the frame transmission is predictable in terms of timing, providing typical reaction times in the order of $200\, ms$.

**CAN**

The Controller Area Network (CAN), was developed in the beginning of the eighties by Bosch$^3$. Today CAN is the most widely used network technology in the automotive industry, found in basically all subsystems defined in Chapter 3. A typical CAN application is any type of embedded system with real-time requirements and loop times of $5 - 50\, ms$, but also non real-time systems are using CAN. Today, CAN holds a strong position where it is used in the automotive application domain and it will continue to be used for a long time. The fact that CAN is such an important network technology for automotive systems (as well as for other application domains) motivates a whole chapter on it. CAN is explained in detail in Chapter 5.

CAN transmits messages in an event-triggered fashion using deterministic collision resolution to control access to the bus (so called CSMA/CR). Messages are transmitted in frames containing 0 to 8 bytes of payload data. These frames can be transmitted at speeds of 10 Kbps up to 1 Mbps, although 500Kbps (e.g., ISO 11898 [77] and SAE J2284 [184]) is the more common choice for powertrain, chassis and non safety-critical x-by-wire systems, whereas speeds of 250 Kbps (e.g., SAE J1939 [183]) are used for control and diagnostics, and 125 Kbps and less (e.g., SAE J1850 [181]) are used typically for body and comfort electronics.

**Byteflight**

Byteflight$^4$ [19] was introduced by BMW in 1996, and then further developed by BMW, ELMOS, Infineon, Motorola and Tyco EC.

The main intended application domain for Byteflight is safety-critical systems, replacing CAN where further development would require higher bandwidth. A typical application is passive safety systems such as airbag systems or seat belt pretensioners with fast response-time requirements and short mission-time. Flexibility, support for event-triggered traffic and higher bandwidth compared with CAN were the main requirements when Byteflight was initially developed. Today, Byteflight is used in the automotive domain by, e.g., BMW.


and in avionic domain by Avidyne. Byteflight is a candidate for future x-by-wire systems, although a more likely candidate is its TDMA extension called FlexRay (described below).

Byteflight is a FTDMA network typically using a star topology (although bus and cluster topologies are also possible). Byteflight guarantees deterministic (constant) latencies for a bounded number of high priority real-time messages. Moreover, it is possible to send low priority non real-time messages in a prioritised manner thanks to the mini-sloting mechanism (described below). Clock synchronisation is provided by a dedicated master node (any Byteflight node can be configured as the master node), achieving clocks synchronised in the order of $100\,\text{ns}$. Another feature of Byteflight is that it is possible to mask babbling idiots using a star coupler.

Messages are scheduled in a cyclic manner using a mini-slotting technique. The essence of the mini-slotting technique is that all nodes in the system keep a slot counter. A specialised node is called the sync master. Any node can be the sync master and it can have redundant backups for fault tolerance. The sync master is initiating a communication cycle periodically by sending a synchronisation pulse (sync pulse). The length/period of a communication cycle is fixed and set to $250\,\mu\text{s}$. The synchronisation pulse is resetting all slot counters to 0. In order for the mini-slotting mechanism to work, to avoid collisions, all messages must have unique identifiers (IDs), as also required when using the CAN network. When a communication cycle is started by the sending of the sync pulse, all slot counters are reset to 0 indicating the start of the cycle’s first minislot. Whenever a node has a message with an ID matching the slot counter, that node will send its message. It will be the only node transmitting a message due to the unique message IDs. Once the message has been sent, all the nodes in the system will detect a new minislot and increase their slot counters by one. However, if there is no message transmitted within a short time $\Delta$ after the initiation of a minislot ($\Delta$ being much shorter than the time needed to transmit a message), a new minislot is detected and all the slot counters are increased again, etc. In this way messages are scheduled in increasing order based on their IDs, serving the lower value IDs first. Hence, a low value ID gives a high priority to a message and thus a higher chance of being transmitted. Note that a minislot can be of varying size depending on the size of the message transmitted in the minislot.

The slot counters might not reach their maximum value (which is 255, i.e., 255 different messages can be served) due to the fact that the maximum length of the communication cycle is reached. As the communication cycle is of a fixed size of $250\,\mu\text{s}$, depending on the number of minislots used (by messages
4.1 Automotive domain

being transmitted) and their individual lengths, the slot counter will only reach a certain value. Therefore some messages might have to wait one or more communication cycles in order to be scheduled (slot counter reaching the message IDs). It should be noted here that the slot counters are always reset to 0 with the reception of a sync pulse.

Babbling idiots is one weak point of a CAN network [22]. This vulnerability to babbling idiots is greatly limited in Byteflight since a node can only transmit a message when its ID matches the slot counter. Hence, a babbling node can transmit at most one message each communication cycle. Using CAN, however, the same node can, if babbling, continuously transmit messages completely starving the rest of the nodes in the system.

One key strength of the Byteflight technology is its support for efficient event-triggered message transmission, although both event- and time-triggered message transmissions are supported. Messages are sent in frames containing 0 to 12 bytes of data (and 4 bytes of overhead). The frames are sent at 10 Mbps using the mini-slotting concept, and the physical medium used is a plastic optical fibre.

MOST

Looking at automotive multimedia and infotainment, the Media Oriented Systems Transport\(^5\) (MOST), is today the de-facto standard to provide communications for multimedia applications.

Initiated in 1997 MOST is intended as a communications network for automotive multimedia applications and has several supporters such as Audi, BMW and Daimler-Chrysler. Typical MOST applications are the interconnection of multimedia and infotainment devices such as video displays, GPS navigation, active speakers and digital radios. Today more than 60 companies are using MOST in their products.

MOST is a master/slave type of network that supports both synchronous and asynchronous traffic with predictable frame transmission at speeds of 25 Mbps. In the near future MOST is expected to be available with speeds of 150 Mbps. MOST is using plastic optical fibre as communication medium, and supports interconnection of up to 64 devices (nodes).

\(^5\)MOST Cooperation: MOST - Media Oriented Systems Transport.
http://www.mostcooperation.com/
Other technologies

LIN, CAN and Byteflight represent strong network technologies with different service possibilities (low, medium and high speed/cost networks), and they are the three more common network technologies used for chassis, passive safety, powertrain, body and comfort electronics, non safety-critical x-by-wire and diagnostics systems. These subsystem classes are historically where the first applications using automotive networking could be found, so many technologies have been developed over the years. Some have been persistent whereas others have been phased out [127, 128, 169]. Some of the other more common technologies are

- **Safe-by-Wire** [21] is a master/slave network mainly used for airbag control. Safe-by-Wire has features taken from CAN and supports communication speeds of 150Kbps. Since CAN and LIN are considered not safe enough for airbag control, the Safe-by-Wire Consortium was formed and developed this network technology.

- **Motorola Interconnect (MI)** [136] is similar to LIN in the sense that it is a simple low-cost master/slave type of network intended for smart sensors in body and comfort electronics such as seats, mirrors and window lifts. However, LIN is close to being the world standard for these types of automotive systems today.

- **Distributed Systems Interface (DSI)** [137] is a master/slave network providing communication speeds of up to 150Kbps, intended for safety-related applications such as airbag systems.

Some of the major automotive networks for multimedia and infotainment, together with MOST, are

- **Domestic Digital Bus (D2B)** [35], by the Optical Chip Consortium, is a ring/star optic network providing communication speeds of up to 20Mbps. D2B was used in some Mercedes Benz models. However, it is worth noticing that Mercedes is using MOST in some of the more recent models.

- **Mobile Multimedia Link (MML Bus)** by Delphi Automotive Systems, is a master/slave optic network providing 100Mbps communications and plug-and-play functionality.

---

6Delphi Automotive Systems: Delphi - Driving Tomorrows Technology. 
http://www.delphi.com/
4.1 Automotive domain

- **IDB-1394 (Automotive Firewire)**\(^7\), was originally used to connect PC devices, but is nowadays also trying to become attractive for the automotive market.

- **USB**\(^8\) is, as Firewire, originally used in the PC market now trying to reach the automotive market.

Summary of current wired network technologies

Table 4.1\(^9\) gives an overview of the automotive network technologies used today, showing which network technologies are used in what applications, with their typical communication requirements and key properties.

Looking at current wired network technologies in the automotive domain, LIN and CAN are the strongest technologies. Also, Byteflight is to some extent used in more safety-critical applications. MOST is the most widely used network technology for multimedia and infotainment systems.

4.1.2 Upcoming wired technologies

X-by-wire systems need fault-tolerant communications with predictable message transmissions and low jitter [45, 222]. This is traditionally solved using TDMA technologies, thanks to their predictable nature. Three of the most common TDMA-based network technologies for automotive applications, namely TTP, TT-CAN and FlexRay, are presented in the following.

**TTP**

The Time-Triggered Protocol (TTP) is part of the Time-Triggered Architecture (TTA) [100] by TTA-Group\(^10\) and TTTech\(^11\). TTA provides time-triggered solutions based on more than 20 years of research on the topic (mainly at TU Wien). TTP was first introduced in 1994 [102] as a pure TDMA protocol nowadays available in two versions, TTP/A [98, 218] and TTP/C [217]. TTP/A is a master/slave TDMA protocol, whereas TTP/C is a fully distributed TDMA protocol more focused towards safety-critical systems, and therefore more complex and expensive than TTP/A. A comparison between LIN and TTP/A can

---

\(^7\)IDB-1394: IDB Forum Homepage. http://www.idbforum.org/

\(^8\)USB.org. http://www.usb.org/

\(^9\)Updated from [143] based on more automotive systems.


be found in [47]. The first TTP off-the-shelf communication controller was released in 1998.

TTP/C is a highly fault tolerant network intended for safety-critical systems such as x-by-wire and avionics. TTP/C implements several fault tolerant mechanisms such as atomic broadcast using membership service, distributed clock synchronisation and bus guardians. TTP/C is very predictable at the cost of being less flexible compared with competing technologies, e.g., FlexRay. However, a valuable property of TTP/C is that it ensures that there can be no single point of failure that can bring the communications to a stop.

Using TTP/C, messages are sent in TDMA rounds. Within the TDMA
round a fixed number of fixed size slots are allocated. All TDMA rounds are of the same duration, and each node in the system is allocated one slot. Here frames are containing 240 bytes of data and 4 bytes of overhead which allows for a high data utilisation rate. Several frames originating from the same node can be transmitted in one slot. It is also possible to transmit messages from different nodes in the same slot using a multiplexing mechanism. Then, however, it must be made sure that two nodes never transmit in the same slot at the same time. The schedule for the TDMA rounds is statically defined and downloaded to all nodes at system start, describing the order and length of the slots, not their contents.

Frames are transmitted at speeds of 5-25 Mbps depending on the physical medium. Research is going on trying to reach speeds of 1 Gbps using an Ethernet based star architecture. There is no physical limitation to the transmission speed as it is when using CAN. TTP/C can be used both with twisted pair and optical medium, and replicated channels are supported for fault tolerance.

Although being excellent from the safety-critical point of view, it seems that TTP/C will not be the choice for automotive x-by-wire applications due to (claimed) limited flexibility, high costs and conflicting interests with the automotive industry [206]. This has led to the initiation of FlexRay (described below). However, in other application domains, such as avionics, TTA is a hot candidate, and TTA is currently further developed in the DECOS project\(^\text{12}\) [103].

**TT-CAN**

The Time-Triggered CAN (TT-CAN)\(^\text{13}\) was introduced in 1999 as a time-triggered layer on top of CAN. TT-CAN is a hybrid TDMA on top of CSMA/CR allowing for both time-triggered and event-triggered traffic. TT-CAN is standardised by ISO [81] and intended for x-by-wire systems. However, it does not provide the same level of fault tolerance as TTP and FlexRay, which are the other two candidates for x-by-wire systems.

TT-CAN is a centralised TDMA protocol in the sense that there is a dedicated node called the time-master responsible for the scheduling and the clock synchronisation of the system. The time-master triggers the start of the schedule and which schedule to use, keeping the nodes synchronised by the time-master’s sending of a specific message called the Reference Message (RM). Using TT-CAN, messages are sent in Basic Cycles (BCs) that contain a fixed

---

\(^{12}\)DECOS: Dependable Embedded Components and Systems. https://www.decos.at/

\(^{13}\)Time-Triggered CAN. http://www.can-cia.de/can/ttcan/
number of fixed (possibly different) length time windows. Hence, a BC is a sequence of fixed length time windows. These windows can be of four different types: reference message, exclusive windows, arbitration windows and free windows. In exclusive windows a pre-defined message is sent. In arbitration windows the standard CAN arbitration mechanism is used so several messages can compete to be sent. Free windows are reserved for possible future expansions of the TT-CAN protocol.

A maximum of 64 different BCs can be allowed, forming what is called a system matrix. However, although the same sequence of time windows in terms of length is repeated for all BCs, the time window type and contents can be different for different BCs. To allow this, each node has to know which BC that is present, and therefore which schedule to use in terms of time window contents. This is possible thanks to a cycle counter. The value of the cycle counter is included in the reference message sent by the time master. Hence, the reference message is providing the start and contents of a BC.

There are two different TT-CAN implementations available called Level 1 and Level 2. Using Level 1 TT-CAN supports only the TDMA scheduling as presented above. Using Level 2 TT-CAN also provides a globally synchronised clock. The clock synchronisation is done using the RM by including a 3 byte global time. Hence, the size of the payload in the RM is 1 byte in Level 1 and 4 bytes in Level 2.

Strong points of TT-CAN are the support of coexisting event- and time-triggered traffic together with the fact that it is standardised by ISO. It is also on top of standard CAN which allows for an easy transition from CAN to TT-CAN. Moreover, there exist off-the-shelf TT-CAN controllers. However, TT-CAN will most likely not be the choice for future x-by-wire systems due to the same limitations as with CAN, i.e., low fault tolerance and low bandwidth.

**FlexRay**

In 1998, BMW and Daimler-Chrysler analysed the current available automotive networks (e.g., CAN, TTP, MOST and Byteflight) and found that none of those technologies fulfil the needs of next generation automotive systems, especially when the automotive industry will take the next step towards x-by-wire.

As a response to this, the FlexRay consortium\(^\text{14}\) was formed with the goal to develop a new network technology called FlexRay [51]. This new network technology should be the solution for the introduction of x-by-wire systems as

\(^{14}\text{FlexRay Consortium: http://www.flexray.com/}\)
well as the replacement of some of the fieldbuses currently used, thus reducing the total number of different in-car network technologies. Today basically all car manufacturers have joined this consortium, and in the middle of 2004 the protocol specification was made public.

FlexRay is a time-triggered extension to Byteflight, providing high speed fault-tolerant communications by combining time-triggered TDMA and the event-triggered friendly FTDMA.

As physical layer, electrical and optical solutions are available relying on either single or dual channels, forming either a passive bus topology, or an active (multiple) star topology. Using single channels between the nodes (which is more common today) reduces the amount of wiring needed and therefore also the cost of the automotive system. On the other hand, dual channels are more expensive, but can tolerate one faulty channel in the case when the same data is continuously sent on both channels. This is desirable in an x-by-wire application, where the highest level of fault tolerance is required. Using dual channels, the same slot can be used by two different nodes, increasing the flexibility of the automotive system in general, providing either doubled bandwidth in the case when different message frames are sent on the two channels in the same slot, or redundancy if one channel would fail. Note that within a dual channel network it is possible to have nodes connected with either a single channel or dual channel.

Messages are sent in frames containing 0 to 254 bytes of payload data and 5 bytes of header. Frames can be either statically scheduled with bounded communication latency, or they can be dynamically scheduled. Statically scheduled frames are sent within the static segment of the communication cycle, and the dynamically scheduled frames are sent within the dynamic segment of the communication cycle. There is no interference between the static and the dynamic segment.

The static segment contains static slots that are used for the transmission of statically scheduled frames. Multiple slots can be allocated for one node in the same communication cycle which allows for intra-cycle application level agreement. The static segment is protected by a bus guardian, preventing babbling idiots.

The dynamic segment contains minislots where dynamically scheduled frames are sent. Here the scheduling is performed in the same way as the Byteflight technology (as presented above). This allows for dynamic bandwidth allocation (also during run-time), either on per node basis or on per channel basis. Note that there is no bus guardian mechanism to protect the dynamic segment from babbling idiots.
FlexRay can be configured in either time-triggered mode or in event-triggered mode. When using the time-triggered mode it is possible to either have a mix of static and dynamic slots or to exclusively have static slots. In time-triggered mode, FlexRay has a built-in clock synchronisation mechanism providing a synchronized time-base with a worst-case deviation between two clocks within $1\mu s$.

In the event-triggered mode there is one static slot and the rest are dynamic slots. Here, one node is a dedicated master that synchronizes the system instead of the distributed clock synchronization used in the time-triggered mode. This master is using the symbol window as the cycle trigger. Trigger monitoring is used to detect errors by observing that the trigger is neither too early nor too late.

As FlexRay is supposed to be first and foremost used in safety-critical applications such as x-by-wire, a lot of work has been done into making FlexRay as fault tolerant as possible. FlexRay provides high robustness against transient faults using a never-give-up strategy. FlexRay nodes are required to be configured to survive several cycles without receiving frames. CRCs are used both in the header part of the frame and for the rest of the frame. The FlexRay bus guardian ensures error containment and error detection in the time domain. If everything is as it should be, the bus guardian enables the bus driver for message transmissions. It also interacts with the host processor to retrieve the schedule and to provide the current status.

FlexRay provides network speeds of up to 10 Mbps and is expected to be the de-facto communication standard for high-speed automotive control applications interconnecting ECUs in future automotive systems. A special area of interest will be high-speed safety-critical automotive systems such as x-by-wire and advanced chassis and powertrain applications.

**Summary of upcoming wired network technologies**

Table 4.2 gives an overview of the upcoming wired automotive network technologies, showing which network technologies are used in what applications, with their typical communication requirements and key properties. Among the presented network technologies, FlexRay is expected to be the de-facto standard for high speed safety-critical communications in the automotive domain.
4.1 Automotive domain

<table>
<thead>
<tr>
<th>Usage</th>
<th>Network technology group</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TTP</td>
</tr>
<tr>
<td>Chassis</td>
<td>SOME</td>
</tr>
<tr>
<td>Passive safety</td>
<td>SOME</td>
</tr>
<tr>
<td>Powertrain</td>
<td>SOME</td>
</tr>
<tr>
<td>Body and comfort</td>
<td>NO</td>
</tr>
<tr>
<td>Multimedia / infotainment</td>
<td>NO</td>
</tr>
<tr>
<td>X-by-wire</td>
<td>YES</td>
</tr>
<tr>
<td>Wireless / telematics</td>
<td>NO</td>
</tr>
<tr>
<td>Diagnostics</td>
<td>SOME</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement</th>
<th>TTP</th>
<th>TT-CAN</th>
<th>FlexRay</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fault tolerance</td>
<td>YES</td>
<td>SOME</td>
<td>YES</td>
</tr>
<tr>
<td>Predictability</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Bandwidth</td>
<td>YES</td>
<td>SOME</td>
<td>YES</td>
</tr>
<tr>
<td>Flexibility</td>
<td>SOME</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Security</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Property</th>
<th>TTP</th>
<th>TT-CAN</th>
<th>FlexRay</th>
</tr>
</thead>
<tbody>
<tr>
<td>Media access</td>
<td>multimaster</td>
<td>multimaster</td>
<td>multimaster</td>
</tr>
<tr>
<td></td>
<td>TDMA</td>
<td>TDMA</td>
<td>TDMA</td>
</tr>
<tr>
<td>Transmission media</td>
<td>twisted pair</td>
<td>twisted pair</td>
<td>twisted pair</td>
</tr>
<tr>
<td></td>
<td>optical</td>
<td>CSMA/CR</td>
<td>FTDMA</td>
</tr>
<tr>
<td>Bit rate (max)</td>
<td>25Mbps</td>
<td>1Mbps</td>
<td>10Mbps</td>
</tr>
<tr>
<td>Data length</td>
<td>240</td>
<td>0-8</td>
<td>0-254</td>
</tr>
<tr>
<td>Cost</td>
<td>H</td>
<td>M</td>
<td>M</td>
</tr>
</tbody>
</table>

Table 4.2: Upcoming wired technologies: usage, requirements and properties.

4.1.3 Wireless technologies

There are several applications pushing for the adoption of wireless communications in automotive systems, both within the vehicle (in-vehicle communications) and between the vehicle and its surroundings (inter-vehicle communications). Looking at in-vehicle communications, more and more portable devices, e.g., mobile phones, portable GSM devices and laptop computers can exploit the possibility of interconnection with the vehicle. Also, several new applications will exploit the possibility of inter-vehicle communications, e.g.,
vehicle-to-vehicle\textsuperscript{15} and vehicle-to-roadside communications. Below, two of the more common wireless technologies that have potential to be used in the automotive domain are presented, namely Bluetooth and ZigBee.

### Bluetooth

Bluetooth (IEEE 802.15.1)\textsuperscript{20, 72} currently provides network speeds of up to 3 Mbps (Bluetooth v2.0). Originally devised for wireless Personal Area Network (PAN) deployment for low-cost, low-power, short-range wireless ad hoc interconnection, the Bluetooth technology has fast become very appealing also for the automotive environment, as a potential automotive wireless network technology.

As a response to this interest, the Bluetooth Special Interest Group (SIG) formed the Car Working Group in December 1999. The Hands-Free profile was the first of several application level specifications expected from the Car Working Group. Using the new Hands-Free profile, products that implement the Bluetooth specification can facilitate automatic establishment of a connection between the car’s hands-free system (typically part of its audio system) and a mobile phone.

The Bluetooth SIG, in November 2004, laid out a three-year roadmap for future improvements to Bluetooth. Prioritised targets include Quality of Service (QoS), security, power consumption, multicast capabilities and privacy enhancements. Long-range performance improvements are expected to increase the range of very low power Bluetooth-enabled sensors to approximately 100 meters.

### ZigBee

ZigBee\textsuperscript{16} (IEEE 802.15.4)\textsuperscript{72, 227} is a new low-cost and low-power wireless PAN standard, intended to meet the needs of sensors and control devices.

Typical ZigBee applications are monitoring and control applications which do not require high bandwidth, but do impose severe requirements on latency and energy consumption. Despite the number of low data rates proprietary systems designed to fulfil the above mentioned requirements, there were no standards that met them. Moreover, the usage of such legacy systems raised


\textsuperscript{16}ZigBee Alliance. http://www.zigbee.org/
4.1 Automotive domain

significant interoperability problems which the ZigBee technology solves, providing a standardized base set of solutions for sensor and control systems. The ZigBee Alliance (with over 120 company members) ratified the first ZigBee specification for wireless data communications in December 2004.

ZigBee provides network speeds of up to 250 Kbps, and is expected to be largely used as a sensor network for monitoring and control purposes (air conditioning, heating, ventilation, lighting control, etc.).

Other technologies

Apart from Bluetooth and ZigBee, two more wireless technologies interesting for the automotive domain are UWB (another wireless PAN) and Wi-Fi (a Wireless Local Area Network (WLAN))

• **UWB**\(^\text{17}\) (IEEE 802.15.3a), or Ultra Wide Band [72], is a potential competitor to the IEEE 802.11 standards, providing speeds of up to hundreds of Mbps and robust communications thanks to its usage of a spread spectrum of frequencies. UWB is likely to be used in applications requiring high bandwidth, such as interconnection of multimedia devices. Other potential automotive applications which could be supported by UWB are collision-detection systems and suspension systems that respond to road conditions. However, UWB being a young technology, these applications are not available yet.

• **Wi-Fi** stands for wireless fidelity and is the general term for any type of IEEE 802.11 network [71].

Wi-Fi can be found in the automotive domain used for inter-vehicle communications by, e.g., the Car2Car Consortium\(^\text{18}\), a non-profit organisation initiated by European vehicle manufacturers. Applications here are advanced drive assistance to reduce the number of accidents, decentralized floating car data to improve local traffic flow and efficiency, and user communications and information services for comfort and business applications to driver and passengers. Research projects working in this area are, for example, the European Network-on-Wheels (NoW) project\(^\text{19}\) and CarTALK 2000\(^\text{20}\), a European research project started in 2001.

\(^{19}\)NOW: Network on Wheels. http://www.network-on-wheels.de/
Wireless technology comparison

A summarising comparison of the above mentioned wireless technologies is presented in Table 4.3.

From a general perspective, the main differences between the wireless technologies considered originates from the different target applications they are optimized for. Bluetooth addresses voice applications, eliminating short distance cabling, is suitable for hands-free audio but also for synchronization of cell phones to PDAs, file transfer, ad-hoc networking between capable devices. For these applications a network range of a few tens of meters is sufficient together with network speeds of a few (1-3) Mbps.

ZigBee, on the other hand, addresses sensors and control, and other short message applications. ZigBee applications are consisting of lots of devices typically requiring small data packets with a lightweight technology and a small protocol stack. Network speed here is not as important as for the other wireless technologies presented in this chapter, and currently only 250 Kbps is provided. Nodes can be scattered around in a slightly larger area compared with Bluetooth.

### Table 4.3: Wireless technologies: technology comparison.

<table>
<thead>
<tr>
<th>Technology</th>
<th>Bluetooth</th>
<th>ZigBee</th>
<th>UWB</th>
<th>Wi-Fi</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard</td>
<td>IEEE 802.15.1</td>
<td>IEEE 802.15.4</td>
<td>IEEE 802.15.3a</td>
<td>IEEE 802.11a</td>
</tr>
<tr>
<td>Freq. band</td>
<td>2.4 GHz</td>
<td>2.4 GHz</td>
<td>3.1-10.6 GHz</td>
<td>2.4 GHz (b/g)</td>
</tr>
<tr>
<td>Network</td>
<td>P2P</td>
<td>Mesh</td>
<td>P2P</td>
<td>P2P</td>
</tr>
<tr>
<td>Modulation technique</td>
<td>Frequency Hopping Spread Spectrum (FHSS)</td>
<td>Direct Sequence Spread Spectrum (DSSS)</td>
<td>Orthogonal Frequency Division Multiplexing (OFDM) or Direct Sequence Spread Spectrum (DSSS)</td>
<td>Orthogonal Frequency Division Multiplexing (OFDM) or Direct Sequence Spread Spectrum (DSSS)</td>
</tr>
<tr>
<td>Modulation technique</td>
<td>1 Mbps (ver 1.0)</td>
<td>250 Kbps</td>
<td>20-100 Mbps</td>
<td>54 Mbps (802.11a)</td>
</tr>
<tr>
<td>Modulation technique</td>
<td>3 Mbps (ver 2.0)</td>
<td>MHz within short ranges expected)</td>
<td>Mbps within short ranges expected)</td>
<td>11 Mbps (802.11b)</td>
</tr>
<tr>
<td>Network range</td>
<td>Up to 100 meters, depending on radio class (effective 10 meters)</td>
<td>Up to 70 meters (effective 20 meters)</td>
<td>Up to 20 meters (effective 10 meters)</td>
<td>Up to 100 meters (effective 50 meters)</td>
</tr>
</tbody>
</table>
4.1 Automotive domain

UWB is the upcomer (although historically it has its roots in the sixties), providing interestingly high network speeds together with a robust communications using a broad spectrum of frequencies. This technology is best suited at very short range (a few meters), compared with the others, but the bandwidth it provides (up to 480 Mbps) is magnitudes higher compared to the other technologies.

Wi-Fi is developed as a replacing technology for wired Ethernet used mainly in home and office environments. To provide mobility, network speeds and range should be as high as possible. 54 Mbps is provided and the network is still effective around 50 meters.

Considering power usage, both ZigBee and UWB require very low power for operation. On the other hand, although much better than Wi-Fi (which is not built with low power as the prime target), Bluetooth requires about 50 times more energy to transfer a single bit compared to UWB.

From a real-time point of view, most telematics applications do not feature real-time requirements in the strict sense. Navigation and traffic information systems require position and Internet-like communications, providing traffic information and directions. Voice applications have slightly higher requirements on QoS, e.g., real-time voice processing and recognition. However, some safety-systems do have real-time requirements, e.g., communications between the vehicle and other vehicles or roadside objects, implementing collision detection/avoidance systems or active suspension systems that respond to road conditions. Moreover, diagnostics and service tools could make real-time data available during operation of the vehicle. Also, real-time requirements are put by the usage of wireless technologies as a redundant link between nodes linked with wired type of networks.

Looking at the potential automotive usage, features from all four technologies are summarised and presented in Table 4.4.

Summary of wireless network technologies

Table 4.5 gives an overview of the upcoming wireless automotive network technologies, showing which network technologies are used in what applications, with their typical communication requirements and key properties.

4.1.4 Discussion on automotive network technologies

In a modern automotive system there is a need for several network technologies to co-exist, e.g., both time-triggered and event-triggered traffic have to be
supported, both high and low speed communications have to be provided, and both low and high cost network technologies have to be available allowing for a fair trade-off between cost and performance.

CAN is very good in handling event-triggered traffic, and today it is the most widely used wired network technology in the automotive domain. CAN has also been extended with a time-triggered layer, called TT-CAN, to support time-triggered traffic. However, since TT-CAN relies on the CAN lower layers, it is lacking fault-tolerant mechanisms and bandwidth capabilities. Hence, CAN/TT-CAN is not commonly suggested for future applications, such as safety-critical x-by-wire systems.

On the other hand, TTP is a highly fault-tolerant network developed and intended for safety-critical systems such as x-by-wire and avionics. However, a drawback with TTP is that, due to its somewhat inflexible message transmission and high cost, the use of another fieldbus is needed for other applications in the car where high bandwidth together with event-triggered capabilities is needed.

While TTP does not directly support event-triggered traffic, FlexRay does, as it combines TDMA message transmission and the FTDMA of Byteflight, thus allowing for both time-triggered and event-triggered message transmis-

<table>
<thead>
<tr>
<th>Technology</th>
<th>Bluetooth</th>
<th>ZigBee</th>
<th>UWB</th>
<th>Wi-Fi</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard</td>
<td>IEEE 802.15.1</td>
<td>IEEE 802.15.4</td>
<td>IEEE 802.15.3a</td>
<td>IEEE 802.11a, IEEE 802.11b, IEEE 802.11g</td>
</tr>
<tr>
<td>Weak points</td>
<td>✓ Interference with Wi-Fi. ✓ Consume medium power. ✓ Low bandwidth. ✓ Short range. ✓ Interference. ✓ Traditionally consume high power.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4.4: Wireless technologies: potential automotive usage comparison.
4.1 Automotive domain

Table 4.5: Upcoming wireless technologies: usage, requirements and properties.

<table>
<thead>
<tr>
<th>Usage</th>
<th>Wireless</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Chassis</td>
<td>Bluetooth</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>Passive safety</td>
<td>Bluetooth</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>Powertrain</td>
<td>Bluetooth</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>Body and comfort</td>
<td>Bluetooth</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>Multimedia / infotainment</td>
<td>Bluetooth</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>X-by-wire</td>
<td>Bluetooth</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>Wireless / telematics</td>
<td>Bluetooth</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Diagnostics</td>
<td>Bluetooth</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Bluetooth</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Fault tolerance</td>
<td>Bluetooth</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>Predictability</td>
<td>Bluetooth</td>
<td>SOME</td>
<td>SOME</td>
</tr>
<tr>
<td>Bandwidth</td>
<td>Bluetooth</td>
<td>SOME</td>
<td>SOME</td>
</tr>
<tr>
<td>Flexibility</td>
<td>Bluetooth</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Security</td>
<td>Bluetooth</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Property</th>
<th>Bluetooth</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Media access</td>
<td>Bluetooth</td>
<td>master/slave</td>
<td>slotted/unslotted</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TDMA</td>
<td>CSMA/CA</td>
</tr>
<tr>
<td>Bit rate (max)</td>
<td>Bluetooth</td>
<td>3Mbps</td>
<td>250Kbps</td>
</tr>
<tr>
<td>Cost</td>
<td>Bluetooth</td>
<td>H</td>
<td>M</td>
</tr>
</tbody>
</table>

Moreover, FlexRay was developed with safety-critical applications in mind, just like TTP. Hence, using FlexRay it is possible to develop a wide range of systems, reducing the need for several fieldbus technologies.

Among TT-CAN, TTP and FlexRay, the latter has the biggest potential for becoming the next generation automotive network for safety-critical fault-tolerant applications, mainly because it is heavily backed up by industrial partners and pushed by most major industrial automakers.

Looking at wireless technologies, Bluetooth is the most widely-used in-car wireless technology today. In a Bluetooth-enabled vehicle, the car audio system takes over the phone function and any Bluetooth device can easily connect to another one (i.e. CD, DVD, MP3 players get access to the car speakers).
Moreover, through Bluetooth interfaces, hand-held computers and diagnostic equipments can interface to the car and access services provided by the on-board diagnostic and control systems. The frequency hopping modulation technique is very suitable to harsh environments often found in automotive applications.

On the other hand, in the automotive context, ZigBee, thanks to its low power and low-latency features, is an alternative for wireless communications in non bandwidth-greedy monitoring and control applications, related to air-conditioning and lighting control, telemetry, vehicle immobilizers, toll collection, vehicle identification, tire tracking/monitoring.

However, for embedded in-vehicle communications, the practical use of wireless solutions can be discussed. As the vehicle is of limited size it is probably possible to connect most nodes with a cable. Also, the wireless nodes require some kind of power supply. If this power supply is to rely on wires, then there is no reason not to use these wires for the communications as well. Moreover, if the wireless nodes have their own battery, this battery is required to work for 15 years as well as in operating temperatures down to -30 degrees Celsius. An alternative is a built in power generator. However, these "wireless power" solutions are costly and remove some of the potentials for wireless sensor networks in automotive systems.

4.2 Other domains

Looking at other application domains, many network technologies can be found. Below, some of the major network technologies found in the avionic, train, industrial and process automation, and building and home automation domains are briefly presented.

4.2.1 Avionic domain

For avionic and aerospace communication systems, the ARINC 429 [8] standard and its newer ARINC 629 [8, 135] successor are the most common communication systems used today. Also, the MIL-1553 standard (developed for the U.S. Air Force and the U.S. Navy) [64] has been used for more than 30 years. Apart from ARINC 629, ARINC 429 and MIL-1553, TTP/C, Byteflight and CAN are used in the avionic domains as well.

ARINC 629 supports both periodic and sporadic communications. The network is scheduled in bus cycles, which in turn are divided in two parts. In
the first part periodic traffic is sent, and in the second part the sporadic traffic is sent. The arbitration of messages is based on mini-slotting. Network speeds are as high as 2 Mbps.

MIL-1533 supports network speeds of 1 Mbps.

4.2.2 Train domain

In the train domain, the Train Communication Network (TCN) [69, 70, 94] and WorldFIP\(^21\) [68, 207, 208] are more common. However, also TTP/C and CAN are used to some extent. The usage of Industrial Ethernet has been discussed, although there is currently not an urgent need as most requirements are fulfilled by today’s network technologies, and Industrial Ethernet is not mature enough [93].

The TCN is widely used in trains, and it is implementing the IEC 61275 standard as well as the IEEE 1473 standard. TCN is composed of two networks: the Wire Train Bus (WTB) and the Multifunction Vehicle Bus (MVB). The WTB is the network used to connect the whole train, i.e., all vehicles of the train. Network data rate is up to 1 Mbps. The MVB is the network used within one vehicle. Here the maximum data rate is 1.5 Mbps.

Both the WTB and the MVB are scheduled in cycles called basic periods. Each basic period consists of a periodic phase and a sporadic phase. Hence, there is a support for both periodic and sporadic type of traffic. The difference between the WTB and the MVB (apart from the data rate) is the length of the basic periods (1 or 2\(ms\) for the MVB and 25\(ms\) for the WTB).

The WorldFIP is a popular communication network in train control systems. WorldFIP is based on the Producer-Distributor-Consumers (PDC) communication model, and the WorldFIP network technology defines an application layer that includes PDC- and messaging-services. Currently, network speeds are as high as 5 Mbps (using fibre optics).

4.2.3 Industrial and process automation domain

In the industrial and process automation domain, which is a domain subject to fieldbus development for a long time, PROFIBUS\(^22\) [68, 89, 166], PROFINet [68, 164], INTERBUS [68, 86] and WorldFIP are together today the stronger technologies. Also, as in the other domains, TTP/C, CAN and CAN-based

\(^{21}\)WorldFIP Fieldbus. http://www.worldfip.org

\(^{22}\)PROFIBUS International. http://www.profibus.com
higher layer protocols [34, 155] can be found as well as Ethernet-based solutions in general, and ZigBee as a wireless alternative.

PROFIBUS is the market leader in this area, used in industrial and process automation, and robotics. PROFIBUS provides master-slave communication together with token mechanisms, and is available with data rates up to 12 Mbps. There are three different versions of PROFIBUS:

1. PROFIBUS - DP (decentralized peripherals) is optimised for speed and low cost.
2. PROFIBUS - PA is designed for process automation.
3. PROFIBUS - FMS is a general purpose version of PROFIBUS.

PROFInet is the open standard for industrial and process automation based on Industrial Ethernet, and allows for existing PROFIBUS solutions to be integrated with PROFInet without modification. Using PROFInet it is possible to get very high performance, in terms of timeliness, with network speeds are as high as 100 Mbps and a jitter accuracy of 1 $\mu$s (using isochronous real time).

### 4.2.4 Building and home automation domain

In the building and home automation domain, two examples of the more common fieldbuses are LonWorks [126] and X10.\(^\text{23}\)

LonWorks is a flexible network technology intended to be used in any application where the usage of a fieldbus could potentially be of benefit.

X10 is a low speed network technology relying on power-line communications which means that no special cables have to be laid in order to provide communications, which overcomes many of the issues with wired networks in building and home automation. High speed power-line communications for building and home automation are driven by the HomePlug [67] alliance.

Apart from LonWorks and X10 there also exists some special purpose network technologies, used to control, e.g., lightning [4, 178] and environmental controls [178].

Finally, as a wireless solution ZigBee is a strong candidate. Interested readers are referred to [92, 96] for more information on building and home automation networks.

\(^{23}\)http://www.x10.org
4.3 Summary

This chapter presents a state-of-practice (SOP) overview of network technologies used by distributed embedded real-time systems. A focus is given to the automotive application domain, presenting also the latest technology developments. However, the avionic, train, industrial and process automation, and building and home automation domains are briefly presented as well.

The goal of this chapter is to give an overview of the network technologies used by distributed embedded real-time systems today, identifying key network technologies used by various applications, present their properties and attributes.

A finding of the overview is that CAN is the most used network technology in the automotive domain, and it is used in many other application domains as well. Hence, a special attention is given to CAN in Chapter 5, presenting details of CAN.
Chapter 5

The Controller Area Network (CAN)

CAN\textsuperscript{1} is one of the major network technologies used in many application domains requiring embedded communications. It is particularly important in the automotive domain. This chapter goes into detail, presenting CAN, its history, its properties, and several higher layer protocols developed for various applications.

5.1 History

In the beginning of the 80s, Robert Bosch GmbH evaluated existing serial bus systems (network technologies) in terms of usability in the automotive domain. None of the existing technologies were suitable, and in 1983 Bosch proposed the Controller Area Network (CAN). This new network technology was intended, primarily, to support adding of new functionality in automotive systems. Moreover, replacing dedicated cables with a shared network also reduce the growing issue of cabling in a vehicle. In February 1986 Bosch presented "Automotive Serial Controller Area Network” at the SAE congress in Detroit, and CAN was officially born. The following year, the first CAN communications adapter chips were released by Intel and Philips. However, it was not until the beginning of the 90s that Bosch submitted the CAN specification for

\textsuperscript{1}Robert Bosch GmbH, BOSCH’s Controller Area Network, http://www.can.bosch.com/
The Controller Area Network (CAN)

Table 5.1: CAN standards and their key differences.

<table>
<thead>
<tr>
<th>Property</th>
<th>ISO 11898</th>
<th>SAE J1939</th>
<th>SAE J1850</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transmission media</td>
<td>Twisted pair</td>
<td>Twisted pair</td>
<td>Twisted pair or single wire</td>
</tr>
<tr>
<td>Bit rate</td>
<td>10 Kbps to 1 Mbps</td>
<td>250 Kbps</td>
<td>10.4 Kbps or 41.6 Kbps</td>
</tr>
<tr>
<td>Header length</td>
<td>11 or 29 bits</td>
<td>29 bits</td>
<td>32 or 8 bits</td>
</tr>
<tr>
<td>Data length</td>
<td>0-8 bytes</td>
<td>8 bytes (typical)</td>
<td>0-8(10) bytes</td>
</tr>
<tr>
<td>Message overhead</td>
<td>9.9%-22%</td>
<td>9.9%-22%</td>
<td>8.3%-33.3%</td>
</tr>
<tr>
<td>Max bus length</td>
<td>40 meters</td>
<td>40 meters</td>
<td>35 meters</td>
</tr>
<tr>
<td>Max no. nodes</td>
<td>32 (typical)</td>
<td>30 or 10</td>
<td>32</td>
</tr>
</tbody>
</table>

international standardisation. In the end of 1993, CAN was standardised by ISO as ISO standard 11898 [77]. At the same time, a fault tolerant version of CAN was standardised as ISO 11519-2 [74]. Two years later, ISO 11898 was extended by an addendum to also include an extended version of CAN (see details below).

Looking at CAN today, ISO 11898 is the most commonly used fieldbus in the European automotive industry. In the US, however, SAE J1850 [181] is more common, although being replaced [128] by SAE J2284 [184]. Also, for trucks and trailers, the SAE J1939 [183] is used since the end of the 90s. The SAE J1939 was published by SAE in 1998, as a result of the work initiated in the beginning of the 90s by the SAE truck and bus control and communications sub-committee. J1939 specifies how messages are defined for engine, transmission and brake systems in truck and trailer applications. Nowadays, SAE J1939 is widely used in truck and trailer applications, and standardised as ISO 11992.

The major features and properties of the above mentioned CAN standards are presented in Table 5.1 [128].

5.1.1 Standards

Over the years, several revised CAN specifications have been standardised. The ISO 11898 standards are

- ISO 11898 - defines the CAN protocol [77].
5.1 History

- ISO 11898-1 - defines the CAN data link layer [78].
- ISO 11898-2 - defines the non-fault-tolerant CAN physical layer [79].
- ISO 11898-3 - defines the fault-tolerant CAN physical layer [80].
- ISO 11898-4 - defines the time-triggered version of CAN [81].
- ISO 11519-2 - defines a low speed fault-tolerant CAN [74].

The SAE J1939 standards are

- J1939/11 - defines the physical layer (250 kbps, shielded twisted pair).
- J1939/12 - defines the physical layer (twisted quad of wires and active bus termination).
- J1939/13 - defines an off-board diagnostic connector.
- J1939/15 - defines the reduced physical layer (250 kbps, unshielded twisted pair).
- J1939/21 - defines the data link layer.
- J1939/31 - defines the network layer.
- J1939/71 - defines the vehicle application layer.
- J1939/73 - defines application layer diagnostics.
- J1939/81 - defines network management.

The ISO 11992 standards are

- ISO 11992-1 - defines the physical and data-link layers [82].
- ISO 11992-2 - defines application layer for brakes and running gear [83].
- ISO 11992-3 - defines application layer for equipment other than brakes and running gear [84].
- ISO 11992-4 - defines diagnostics [85].
Finally, looking at other application domains, for tractors and machinery for agriculture and forestry, an SAE J1939-based ISO standard is used: ISO 11783 [75, 76], and NMEA 2000© [140] defines a SAE J1939/ISO 11783 based protocol for marine usage.

In the remainder of this thesis, all above mentioned groups of CAN standards are treated as “CAN” for simplicity, as their differences mainly are in speed and usage of message identifiers.

5.2 Applications

A typical CAN application is any type of embedded system with real-time requirements and cycle times of $5 - 50\, ms$. However, CAN is used for many non real-time applications as well.

CAN was first used in the automotive industry by Mercedes-Benz in 1992. Initially, one CAN bus was used for engine control, but as a second step, a gateway was introduced connecting the engine control network with a new CAN network controlling body and comfort electronics. For a good overview of applications where CAN is used interested readers are referred to the CAN in Automation (CiA) website.

5.3 Technical properties

CAN is a broadcast bus, which uses deterministic collision resolution to control access to the bus (so called Carrier Sense Multiple Access / Collision Resolution, CSMA/CR). CAN transmits messages in an event-triggered fashion using frames containing 0 to 8 bytes of payload data. These frames can be transmitted at speeds of 10 Kbps up to 1 Mbps.

5.3.1 Frame arbitration

The CAN message frame identifier is required to be unique, in the sense that two simultaneously active frames originating from different sources must have distinct identifiers. Besides identifying the frame, the identifier serves two purposes: (1) assigning a priority to the frame, and (2) enabling receivers to filter frames.

---

2CAN in Automation (CiA). http://www.can-cia.org/
The basis for the access mechanism is the electrical characteristics of a CAN bus. During arbitration, competing communication adapters simultaneously put their identifiers, one bit at the time, on the bus. Bit value “0” is the dominant value. Hence, if two or more communication adapters are transmitting bits at the same time, and if at least one communications adapter transmits a “0”, then the value of the bus will be “0”. By monitoring the resulting bus value, a communications adapter detects if there is a competing higher priority frame (i.e., a frame with a numerically lower identifier) and stops transmission if this is the case. Because identifiers are unique within the system, a communications adapter transmitting the last bit of the identifier without detecting a higher priority frame must be transmitting the highest priority active frame, and can start transmitting the body of the frame, i.e., CAN implements CSMA/CR. Thus, CAN behaves as a global priority based queue, i.e., a fixed priority non pre-emptive system. This since at all communication adapters (nodes) the message chosen during arbitration is always the active message with the highest priority. Globally, the message with the highest priority will always be selected for message transmission.

5.3.2 Topology

CAN is normally used as a bus, i.e., bus topology. However, star topologies are also possible [13, 14].

5.4 Scheduling of CAN

As presented in Chapter 2, schedulers can be divided into three groups: time-driven, priority-driven and share-driven schedulers.

5.4.1 Priority-driven

For CAN, priority-driven is the most natural scheduling method since Fixed Priority Scheduling (FPS) is the policy implemented by the CAN arbitration mechanism. Analysis have been presented to determine the schedulability of CAN message frames [211, 213, 214]. This analysis is based on the standard FPS response-time analysis for CPU scheduling [9].
5.4.2 Time-driven

By providing some higher layer protocol running on top of CAN, it is possible to achieve time-driven scheduling. These protocols implement a master/slave mechanism, having a central master node controlling the network in a time-driven fashion. An example of a time-driven scheduler for CAN is TT-CAN [81]. Also, FTT-CAN [5, 6] provides time-driven scheduling as well as an option to combine time-driven and priority-driven scheduling. TT-CAN and FTT-CAN are presented in more detail in Section 5.6.

5.4.3 Share-driven

By providing the option of share-driven scheduling of CAN, designers are given more freedom in designing a distributed real-time system where the nodes are interconnected with a CAN network. As time-driven schedulers, share-driven schedulers are using a master/slave mechanism running on top of CAN, scheduling the network according to share-driven scheduling policies. Server-CAN [141, 148, 152] is a share-driven scheduler for CAN presented in more detail in Section 5.6 and chapters 6 to 9.

5.5 Schedulability analysis

In general, as presented in Section 2.4, utilisation based tests for non-preemptive systems with blocking can be used to determine the schedulability of a CAN network. However, over the years, to determine the schedulability of CAN based systems, focus has been on developing response-time tests.

5.5.1 Classical response time analysis

Tindell et al. [211, 213, 214] present analysis to calculate the worst-case latencies of CAN frames. This analysis is based on the standard FPS response-time analysis for CPU scheduling presented by Audsley et al. [9] (Section 2.4.5).

Calculating the worst-case response-times requires a bounded worst-case queuing pattern of frames. The standard way of expressing this is to assume a set of traffic streams, each generating frames with a fixed priority. The worst-case behaviour of each stream, in terms of network load, is to send as many frames as they are allowed, i.e., to periodically queue frames in their corresponding communications adapter. In analogue with CPU scheduling, a model with a set $S$ of streams (corresponding to message transmitting CPU tasks) is
5.5 Schedulability analysis

used. Each $S_i \in S$ is a triple $< P_i, T_i, C_i >$, where $P_i$ is the priority (defined by the message frame identifier), $T_i$ is the period, and $C_i$ is the worst-case transmission time of message $i$. The worst-case latency $R_i$ of a CAN frame sent on stream $S_i$ is, with the assumption of a minimum variation in queuing time relative to $T_i$ to be 0, defined by

$$R_i = J_i + q_i + C_i$$  \hspace{1cm} (5.1)

where $J_i$ is the queuing jitter of the frame, i.e., the maximum variation in queuing-time relative to the start of $T_i$, inherited from the sender task which queues the frame, and $q_i$ represents the effective queuing-time, given by

$$q_i^n = B_i + \sum_{j \in hp(i)} \left( \frac{q_i^{n-1} + J_j + \tau_{bit}}{T_j} \right) \times (C_j + 3 \times \tau_{bit})$$  \hspace{1cm} (5.2)

where

- $C_j$ is the transmission time of message $j$. How to calculate $C_j$ is presented below.
- $\tau_{bit}$ (the bit-time) caters for the difference in arbitration start-times at the different nodes due to propagation delays and protocol tolerances.
- $hp(i)$ is the set of messages with priority higher than that of message $i$.
- $lp(i)$ is the set of messages with priority lower than that of message $i$.
- $3 \times \tau_{bit}$ represents the inter-frame space. Traditionally, Tindell et al. [211, 213, 214] considers the inter-frame space as a part of the data frame, but separating it removes a small source of pessimism in the equations, as pointed out by Broster et al. [23].
- $B_i = \max_{k \in lp(i)} (C_k) + 3 \times \tau_{bit}$ is the worst-case blocking-time of frames sent on $S_i$. Note that as identifiers are required to be unique, there are no blocking message frames with priority equal to the message frame under analysis.

Note that Equation 5.2 is a recurrence relation, where the approximation to the $(n+1)$th value is found in terms of the $n$th approximation, with the first approximation set to $q_i^0 = 0$. A solution is reached either when the $(n+1)$th value is equal to the $n$th, or when $R_i$ exceeds its message deadline or period. The recurrence relation will always terminate given that the total bus utilization is $\leq 1$ [193, 196], i.e., $\sum_{S_i \in S} \left( \frac{C_i + 3\tau_{bit}}{T_i} \right) \leq 1$. 
The Controller Area Network (CAN)

Bits exposed to bit-stuffing (34 control bits and 0-8 bytes of data → 34-98 bits)

- Arbitration field
- Control field
- Data field
- CRC field

<table>
<thead>
<tr>
<th>Arbitration field</th>
<th>Control field</th>
<th>Data field</th>
<th>CRC field</th>
</tr>
</thead>
<tbody>
<tr>
<td>11-bit identifier</td>
<td>4-bit DLC</td>
<td>0-8 bytes</td>
<td>15-bit CRC</td>
</tr>
<tr>
<td>0</td>
<td>000</td>
<td>11111111</td>
<td>Int</td>
</tr>
<tr>
<td></td>
<td></td>
<td>01111111</td>
<td>End of frame</td>
</tr>
</tbody>
</table>

Known bit-values (standard format data frame)

Figure 5.1: CAN frame layout (standard format data frame).

Message transmission time

In CAN, six consecutive bits of the same polarity (111111 or 000000) are used for error and protocol control signalling. To avoid these special bit-patterns in transmitted frames, a bit of opposite polarity is inserted after five consecutive bits of the same polarity. By reversing the procedure, these bits are then removed at the receiver side. This technique, which is called bit-stuffing, implies that the actual number of transmitted bits may be larger than the size of the original frame, corresponding to an additional transmission delay which needs to be considered in the analysis.

The number of bits, beside the data part in the frame, that are exposed to the bit-stuffing mechanism are defined as \( g \in \{34, 54\} \). This since there are either 34 (CAN standard format) or 54 (CAN extended format) bits (besides the data part of the frame), which are exposed to the bit-stuffing mechanism. 10 bits in the CAN frame are not exposed to the bit-stuffing mechanism (see Figure 5.1).

The number of bytes of data in a CAN message frame \( i \) are defined as \( L_i \in [0, 8] \). Recall that a CAN message frame can contain 0 to 8 bytes of data. Hence, the total number of bits in a CAN frame before bit-stuffing is

\[
g + 10 + 8 \times L_i \tag{5.3}
\]

where 10 is the number of bits in the CAN frame not exposed to the bit-stuffing mechanism. Since only \( g + 8L_i \) bits in the CAN frame are subject to bit-stuffing, the total number of bits after bit-stuffing can be no more than

\[
g + 10 + 8 \times L_i + \left\lfloor \frac{g + 8 \times L_i - 1}{4} \right\rfloor \tag{5.4}
\]

Intuitively the above formula captures the number of stuff-bits in the worst-case scenario, shown in Figure 5.2.
Let $\tau_{\text{bit}}$ be the worst-case time taken to transmit a bit on the bus – the so-called bit-time. Hence, the worst-case time taken to transmit a given frame $i$ is

$$C_i = (g + 10 + 8 \times L_i + \left\lfloor \frac{g + 8 \times L_i - 1}{4} \right\rfloor) \times \tau_{\text{bit}}$$  \hspace{1cm} (5.5)$$

### 5.5.2 Extensions to classical analysis

Several extensions to the classical response-time analysis for CAN have been presented in the research community over the years. This section briefly presents a handful of these extensions.

#### Probabilistic analysis

The classical analysis uses a model of CAN messages with the worst-case message length. By using a model where the CAN frame length is described using a distribution of stuff-bits [147] instead of a worst-case value (as done using Equation 5.4), pessimism is reduced, allowing for trade-offs regarding timeliness and reliability. Based on this, analysis techniques have been presented providing a probabilistic worst-case response time [145, 146].

By also considering reliability, there is always some small probability of system failure, regardless of how rigor the schedulability analysis is made, i.e., it is fair to say that “there is no such thing as a hard real-time system”.

Reliability is defined as the probability that a system can perform its intended function, under given conditions, for a given time interval. Missing a deadline in a hard-real time system is, just as a hardware failure, a violation of the intended behaviour. Hence, the probability of missing a deadline is for such a system an important aspect to consider when calculating the overall system reliability. The aspects that need to be considered in calculating the
overall reliability, e.g., for a communication system (like CAN), are depicted in Figure 5.3.

The presented link between timing guarantees and reliability forms a basis for making trade-offs between the two, i.e., some deadline misses could be allowed as long as their effect on the system reliability does not invalidate the overall system reliability requirement.

**Fault models**

In order to handle faults, the original analysis can be extended to handle the effect of errors in the channel. However, the fault model used by Tindell et al. [213] is very simple and thus not really appropriate to describe real faults. Only strictly periodic interference is modelled. An extension to the original fault model is presented by Punneckat et al. [167]. Here faults can be modelled as specific patterns. Basically, periodic bursts and multiple sources of interference can be modelled.

The above mentioned fault models are based on an assumption of minimum inter-arrival time, i.e., bounded number of faults in a specified interval. However, several sources of interference, e.g., electromagnetic interference, are more accurately described as a random pulse train following a Poisson distribution [26]. Trying to represent this using minimum inter-arrival times is not easy. Rather, a probabilistic approach would be more suitable. Navet et al. [138] present a probabilistic fault model, where the faults are described as stochastic processes. These stochastic processes consider both the frequency of the faults and their gravity. Both single-bit faults and bursts of faults can be represented. However, the approach presented by Navet et al. [138] is very pessimistic. Broster et al. [24] present a more accurate probabilistic approach,
5.6 Higher layer protocols

where distributions of worst-case response-times can be obtained when there are probabilistic events in the system, e.g., faults caused by electromagnetical interference [25].

Hansson et al. [62, 63] present a completely different approach. Here the feasibility of the system is determined using simulation. Using simulation even more complex sources of interference can be used, achieving a more realistic result compared to the analytic approaches described above. However, the weakness of using simulation is to determine whether or not the coverage of the simulation is good enough for the considered application.

Reduction of pessimism

Broster et al. [23] propose to reduce the pessimism in the original classical CAN analysis. The inter-frame space (“Int” in Figure 5.1) is separated from the CAN frame in order to remove a small source of pessimism by considering that the CAN frame is available as soon as the last bit of the previous frame is sent, rather than only after the inter-frame space following the completion of the sending of the previous frame.

5.6 Higher layer protocols

Due to its popularity and that CAN only comprise layer 1+2 of the OSI layers [39, 228], several higher layer protocols have been developed on top of CAN over the years, both in the academia as well as commercial ones. These higher layer protocols simplify the usage of CAN in terms of, e.g., development and maintenance, by their capabilities and tool support.

5.6.1 Academic protocols

The reason for having a higher-layer protocol is often to use another scheduling policy than the one offered by CAN. Original CAN is suitable to handle periodic real-time traffic according to the FPS approach. Limiting CAN to periodic traffic, the timing analysis can easily be applied and feasibility checked. However, due to the limitations of FPS scheduling, adaptations to allow other scheduling policies have been done. As an alternative to the fixed-priority mechanisms offered by CAN some higher-layer protocols have been developed to implement priority-driven DPS schedulers such as EDF, time-driven schedulers and share-driven schedulers. Three academic higher layer protocols are briefly presented, namely EDF solutions, FTT-CAN and Server-CAN.
Several methods for dynamic priority scheduling (DPS) scheduling have been proposed for CAN. By manipulating the message identifier, and therefore changing the message priority dynamically, several approaches to mimic EDF type of scheduling have been presented [44, 123, 229]. However, by manipulating the identifier of the CAN frames, these solutions all reduce the number of possible identifiers to be used by the system designers. This could be problematic, since it interferes with other design activities, and is even sometimes in conflict with adopted standards and recommendations [34, 182].

Looking at the approach by Zuberi et al. [229], they propose the usage of a Mixed Traffic Scheduler (MTS), which attempts to give a high utilisation (like EDF) while using CAN’s 11-bit format for the identifier. Using the MTS, the message identifiers are manipulated in order to reflect the current deadline of each message. However, since each message is required to have a unique message identifier, they suggest the division of the identifier field into three sub-fields.

Other suggestions for scheduling CAN according to EDF include the work by Livani et al. [122] and Di Natale [44]. These solutions are all based on manipulating the identifier of the CAN frame, and thus they reduce the number of possible identifiers to be used by the system designers.

Using FTT-CAN (described below), Pedreiras and Almeida [161] show how it is possible to send periodic messages according to EDF using the synchronous window of FTT-CAN. Using their approach, greater flexibility is achieved since the scheduling is not based on manipulating the identifiers. Instead, there is a master node performing the global scheduling of the CAN bus. Moreover, interesting in the context of this thesis is that the paper discusses (and show for FTT-CAN) the level of schedulability possible when scheduling CAN messages according to EDF, i.e., the upper bound on network load when scheduling the messages according to EDF is given. Original CAN implements FPS type of scheduling, where classical task scheduling guarantees a bound of 69% [121], although it has been shown that rate monotonic (RM) scheduled task sets are schedulable with a utilisation requirement of up to 88% on average [110]. RM and EDF have been compared in the context of CAN message scheduling [230] using realistic loads, showing a difference in schedulability of approximately 20%. FTT-CAN has been shown to efficiently schedule message according to EDF with a network load of up to 96% of the theoretical maximum load possible when using FTT-CAN [161], to be compared with the theoretical upper bound of 100% for pre-emptive task scheduling [121].
difference is claimed to originate from FTT-CAN specific mechanisms such as the insertion of idle-time.

**FTT-CAN**

Flexible Time-Triggered CAN (FTT-CAN), presented by Almeida et al., supports priority-driven scheduling in combination with time-driven scheduling. In FTT-CAN, time is partitioned into Elementary Cycles (ECs) that are initiated by a special message, the Trigger Message (TM). This message contains the schedule for the synchronous traffic (time-triggered traffic) that shall be sent within this EC. The schedule is calculated and sent by a specific node called the *master node*. FTT-CAN supports both periodic and aperiodic traffic by dividing the EC in two parts. In the first part, the asynchronous window, a (possibly empty) set of aperiodic messages are sent using CAN’s native arbitration mechanism. In the second part, the synchronous window, traffic is sent according to the schedule delivered in the TM. More details of the EC layout are provided in Figure 5.4, that shows two ECs during which both asynchronous messages (AM\(x\)) and synchronous messages (SM\(x\)) are being sent.

In the figure, the schedule is represented by a bit field, where a “1” in position \(n\) means that synchronous message \(n\) is to be sent during the EC. The asynchronous messages compete for bus access using CAN’s native arbitration mechanism.

As mentioned above, it has also been shown how to schedule periodic messages according to EDF using the synchronous part of the EC [161]. This provides greater flexibility compared to [44, 123, 229], since the scheduling is not based on manipulating the identifiers. Instead, there is a master node performing the scheduling of the CAN bus. Also, a method for calculating the worst-case response-time of the messages using the asynchronous window

![Figure 5.4: EC layout and TM data contents (FTT-CAN approach).](image-url)
have been developed [160], and the fault tolerant operation of FTT-CAN has been investigated [49, 53, 177].

**Server-CAN**

Server-CAN [141, 148, 152] is the share-driven scheduling approach for CAN presented in this thesis. Here, as with FTT-CAN, the network is scheduled by a specialised master node, partitioning time into Elementary Cycles (ECs). These ECs are initiated by a specific message, the Trigger Message (TM). The TM is constructed and sent by the master node. By having a centralised scheduler, various share-driven scheduling policies can be implemented. All details regarding Server-CAN are explained in Chapters 6-9.

Figure 5.5 presents the layout of the EC when using the Server-CAN framework. Note that the N-Servers that are scheduled to transmit a message in the EC are indicated by a ‘1’ in the corresponding position of the bit field in the TM, and that the actual order of transmission is determined by the message identifiers, not by the N-Server number. In the first EC in Figure 5.5 N-Server 8 is allowed to transmit a message. However, in the example the N-Server 8 do not have any messages to transmit.

**Quality of Service**

A common way to send non real-time messages on CAN is to allocate message identifiers with lower priority than all real-time messages. In this way it can be made sure that a non real-time message can block a real-time message at most for the duration of the transmission of one message. However, unwise message identifier assignment to non real-time messages could cause some of them to suffer from starvation. To provide Quality of Service (QoS) for non real-time
messages several approaches have been presented [32, 125]. These approaches dynamically change message identifiers in a way preventing systematic penalisation of some specific messages.

### 5.6.2 Commercial protocols

CAN is heavily used in numerous application domains, and several commercial CAN protocols have been developed over the years. Below, four commercial higher layer protocols are briefly presented, namely TT-CAN, CAN Kingdom, CANopen and DeviceNet.

**TT-CAN**

Time-triggered communication on CAN is specified as TT-CAN, the ISO 11898-4 standard [81], a standardised extension to original CAN. In TT-CAN, the exchange of messages is controlled by the temporal progression of time, and all nodes are following a pre-defined static schedule. One node, the master node, is periodically (or on the occurrence of a specific event) transmitting a specific message, the Reference Message (RM), which acts as a reference in time. All nodes in the system are synchronising with this message, which gives a reference point in the temporal domain for the static schedule of the message transactions. The static schedule is based on a time division scheme, where message exchanges may only occur during specific time slots or in time windows (so called Time Division Multiple Access, TDMA). Hence, the master’s view of time is referred to as the network’s global time.

TT-CAN appends a set of new features to the original CAN, and being standardised, several semiconductor vendors manufacture TT-CAN compliant devices.

**CAN Kingdom**

CAN Kingdom³, introduced by KVASER AB 1991, is a set of protocol primitives based upon ISO 11898 together with a tool, allowing design and efficient message identifier allocation of embedded distributed real-time systems (relying on CAN). A CAN Kingdom based system has one master node. During initiation of a CAN Kingdom based system, a complete higher layer protocol is built up, including data formats, bus management, global clock, identifier assignments etc. During the initialisation phase the master checks which nodes

---

³CAN Kingdom, http://www.cankingdom.org/
that are connected in the system, and assigns proper identifiers to all messages used in the system. All details can be found in the CAN Kingdom specification [55].

**CANopen**

In March 1992, the "CAN in Automation" (CiA) international users and manufacturers group was formed. Later, the same year, they published the "CAN Application Layer" (CAL). CAL was later extended into what is now the CANopen protocol [34], released in 1995. CANopen is designed for motion oriented machine control networks, and can today be found in various application domains, e.g., medical equipment, off-road vehicles, maritime electronics, public transportation, building automation etc. The CANopen standards cover application layer and communication profile, a framework for programmable devices, and recommendations for cables and connectors. The application layer and the CAN based profiles are implemented in software.

**DeviceNet**

Another CAN-based higher layer protocol is DeviceNet. In the beginning of 1994, the DeviceNet specification [155] was released by Allen-Bradley. The development was initiated already in 1992. Today, the DeviceNet standard is developed and maintained by The Open DeviceNet Vendor Association (ODVA)4. DeviceNet was developed mainly for factory automation, where it holds a strong position today.

**5.7 Summary**

This chapter presents all technical details of CAN that are necessary for this thesis. Apart from that, the history of CAN, its birth, major standards and applications are described. It is presented how the scheduling is performed when using standard CAN and analysis techniques for this scheduling are presented. Finally, both academic and commercial higher layer CAN protocols are briefly presented.

CAN is heavily used today in numerous applications, and will continue to be so for a long time. There are no competing network technologies available

---

4The ODVA Official Website, http://www.odva.org/
today in several CAN dominated segments, e.g., as is the case in the automotive domain.

The chapter shows where Server-CAN is located on the "world map of CAN". In the next chapter, details of Server-CAN are presented.
Chapter 6

Share-driven scheduling of CAN

In optimising the design of a real-time communications system, allowing for easy integration of subsystems, it is important to guarantee the timeliness of the subsystem’s execution on the nodes, to minimise their interference (with each other) in the nodes, to guarantee their timely message transmissions, and to minimise the interference between a subsystem’s traffic on the transmission of other traffic sent on the same network.

As Chapter 4 and Chapter 5 have shown, CAN is one of the more important network technologies in the area of embedded real-time communications. From a real-time point of view, the standard CAN MAC protocol implements a bitwise arbitration mechanism that schedules messages according to the FPS policy. This, and the fact that CAN message identifiers represent the fixed priority of a message, makes integration of distributed embedded subsystems relying on CAN a delicate task, as messages belonging to one subsystem affect the temporal performance of other messages and subsystems. Therefore, share-driven scheduling of CAN is proposed [148] as a way to facilitate subsystem integration on the network, jointly scheduling periodic and aperiodic traffic by using servers. This chapter presents a share-driven scheduling framework for standard CAN, i.e., standard CAN communications adapters can be used, as no modifications to the hardware is required. The share-driven scheduling framework is called Server-CAN.
The main contributions of this chapter can be summarised as:

- The concept of Server-CAN is presented and design decisions are discussed, resulting in a generic CAN scheduler (a framework) for implementation of server-based scheduling algorithms (for more details on server-based scheduling algorithms in general, see Section 2.3.3).
- The Server-CAN protocol [141, 149, 152, 154] is presented, explaining Server-CAN’s run-time operation (the scheduler) and protocol messages (the Trigger Message (TM) and the STOP message (STOP)).
- The run-time components of the Server-CAN framework are presented in detail, namely the M-Server, the N-Server and the B-Server.
- The Server-CAN advanced run-time mechanisms are presented. These mechanisms are used to facilitate admission control, mode changes and bandwidth sharing.
- Finally, the fault-tolerant operation of Server-CAN is discussed. However, only initial results in this topic are presented as it is a subject of ongoing research.

All of this together forms the Server-CAN framework.

6.1 Scheduler design

As the objective is to schedule CAN using a share-driven scheduler relying on standard CAN communications adapters, several design decisions have to be made. Looking at standard CAN, the message transmissions are scheduled according to a fixed-priority priority-driven scheduler implemented by the CAN MAC protocol [77]. Other proposals to schedule CAN differently have been realised by either manipulating the CAN message identifiers (i.e., the priorities) [44, 123, 229] or by running a higher layer protocol (a higher layer scheduler) on top of CAN, e.g., TT-CAN [81] and FTT-CAN [5, 6].

6.1.1 Design questions

Two fundamental design questions have to be answered when implementing a share-driven scheduler using standard CAN communications adapters:

- How to implement the scheduler?
6.1 Scheduler design

- Where to implement the scheduler?

6.1.2 How to implement the scheduler

It is possible to realise a share-driven scheduler in a CAN network using any of the above mentioned methods (i.e., either by manipulating the CAN message identifiers or by running a higher layer protocol). However, from an integration point of view, manipulating message identifiers is not an attractive solution as it would require all subsystems to be reconfigured depending on which other subsystems they are to be integrated with. On the other hand, by running a higher layer protocol the message identifiers of the subsystems can be kept (as long as they are unique within the system). Hence, based on these observations, the decision of implementing the share-driven scheduler as a higher layer scheduler is made.

Running a higher layer scheduler on top of standard CAN communications adapters requires a scheduling protocol to be implemented in software, as done with, e.g., FTT-CAN. The overhead caused by the execution of the software scheduler and the overhead on the network caused by the transmission of the scheduler’s protocol messages should both be as small as possible. This is particularly important for CAN, since it is often used in resource constrained applications that have stringent requirements on cost (e.g., in automotive applications).

As standard CAN implements an FPS scheduler which (in the pre-emptive task scheduling case) has a utilisation based schedulability bound of 69% for real-time guarantees (see Section 2.4), running a higher layer protocol would further decrease the available bandwidth in the system (as the protocol messages will consume bandwidth). Instead, scheduling CAN according to a DPS scheduler (e.g., according to EDF) could theoretically raise this bound to 100% (see Section 2.4). Hence, it would be beneficial if the new scheduler could schedule the network according to EDF (and inspiration can be taken from the different EDF schedulers for CAN presented in Chapter 5).

6.1.3 Where to implement the scheduler

There are two alternatives regarding where to implement the higher layer scheduler:

1. A fully distributed scheduler that is running the same (or similar) software on all nodes in the system.
2. A centralised scheduler running the main software on a dedicated node.

As the resources in the system are limited, running a scheduler on all nodes in the system is not always possible (as it would require too much of the system’s total resources). Instead, using a centralised approach gives less impact on the average node in the system, yet providing the scheduling capabilities required. Also, by using a centralised scheduler, normally no message transmissions are required in order to keep the consistency in the system, as could be required by a distributed solution. Moreover, it is easier to implement advanced scheduling algorithms, such as bandwidth sharing algorithms [18, 120], when all system parameters are local to the scheduler.

A drawback of using a centralised scheduler is its vulnerability to failures, i.e., using a centralised approach leads to a single point of failure that in case of failure would have severe impact on the scheduling of the system. However, several mechanisms can be used to improve the fault-tolerant capabilities of a centralised approach (as discussed in Section 6.6).

Running a higher layer scheduler usually requires sending of protocol messages, e.g., the Reference Message (RM) of TT-CAN and the Trigger Message (TM) of FTT-CAN. As CAN’s maximum bandwidth is physically limited to 1 Mbps (with a constraint on the physical length of the network cable due to the propagation time of the bit transmissions over the medium), introducing a high number of protocol messages is not an attractive option. Instead, the number of protocol messages should be small, i.e., the scheduler’s overhead on the network should be low.

6.1.4 Design decisions

Based on the discussions above, a number of design decisions (requirements on the scheduler) have been made:

- The scheduler shall be implemented as a higher layer protocol running on standard CAN communication adapters.

- The scheduler shall schedule the messages according to EDF.

- The scheduler shall be implemented using a centralised approach.

- The scheduler shall require as few protocol messages as possible.
6.2 Server-CAN concept

Using the Server-CAN concept, bandwidth is allocated to *users* of the network. From a network point of view, a subsystem consists of a set of users that are resident in the various nodes comprising the subsystem. A *user* is from a network point of view a stream of messages, e.g., the stream of messages generated by a part of the subsystem, and can be of either periodic or aperiodic nature.

The usage of network access servers called *N-Servers* ensures that no user is consuming more bandwidth than it has been allocated. Each node has one or more N-Servers allocated to it. A typical N-Server *s* has exclusive associated bandwidth $Q_s$, capacity $C_s$, and period time $T_s$. Moreover, all N-Servers have an associated relative deadline $D_s$, and an associated absolute deadline $d_s$, where the former is used by the scheduler when updating the latter, and the latter is used when scheduling an N-Server for message transmission. All N-Servers have a local message queue $q_s$ in which its *user messages* are stored.

Time is partitioned into *Elementary Cycles* (ECs), similar to the FTT-CAN scheduler (see Section 5.6.1), and the length of an EC is denoted $T_{EC}$.

The scheduling of the CAN network is performed at a specialised master node, where a scheduler called the *M-Server* is resident. The M-Server keeps all information regarding the communications in the system, such as all N-Server specifications. Moreover, the M-Server is updating the N-Server parameters during run-time, according to the scheduling policy in use and based on the actual message traffic in the network. As soon as an N-Server is scheduled for message transmission, the N-Server selects a message from its local message queue and sends the message to its local CAN communications adapter. Since each N-Server has exclusive right to a share (fraction) of the total system bandwidth, all users sharing an N-Server will share this fraction of bandwidth. Hence, depending on the type of queue used at the N-Server, e.g., FIFO (First In First Out) or priority ordered, different guarantees (in terms of timeliness) can be offered. Suppose a priority ordered queue is used, in which case the users of that N-Server will experience a service, in terms of timeliness, similar to the one of an exclusive CAN network – essentially with the only difference being the lower bandwidth offered. Hence, the temporal behaviour will, compared to an exclusive CAN network, be divided by the N-Server’s share:

$$Q_s = \frac{C_s}{T_s}$$  \hspace{1cm} (6.1)

In this case a variant of the response-time analysis for fixed-priority systems
Share-driven scheduling of CAN

(described in Section 2.4.5) can be used to calculate the timing properties. This is further discussed and shown in Chapter 9.

The M-Server and all N-Servers are sending their messages to their corresponding CAN communications adapter, where, in turn, all messages are scheduled according to the standard CAN message arbitration mechanism. Hence, looking at the Server-CAN framework, different scheduling policies are performed at different levels as depicted in Figure 6.1. Standard CAN communications adapters can be used as no modifications to the CAN hardware are required.

**Assumption 6.2.1.** In this thesis, all CAN communications adapters are for simplicity assumed to have an infinite message buffer that is ordered based on message identifiers (i.e., message priorities).

The assumption is made as an arbitrary node can have an arbitrary number of N-Servers. Therefore, at an arbitrary node, after the decoding of the TM, an arbitrary number of messages can be sent to the local CAN communications adapter.
6.3 Protocol basics

The Server-CAN framework is used for implementation of server-based scheduling techniques. Using servers, the whole network is scheduled as a single resource, providing bandwidth isolation as well as fairness among subsystems integrated on the network. In the general real-time scheduling theory, a server is a conceptual entity that controls the access to some shared resource. Sometimes multiple servers are associated with a single resource. For instance, in this thesis multiple servers are mediating the access to a single communication resource, the CAN network.

However, in order to make scheduling decisions, share-driven server-based schedulers usually rely on information, such as events, indicating when messages arrive at the different nodes, i.e., when messages are ready for transmission on the network this must be notified to the scheduler in some way. For single processor task scheduling this kind of information is easily provided to the task scheduler. However, in a distributed system this kind of information has to be communicated using message transmissions. The information about message readiness at the nodes is needed by the server scheduling mechanism in order to assign proper deadlines (according to the scheduling policy in use) that are to be used by the EDF scheduler. However, as explained in Section 6.1, it is not desirable that the information on when messages arrive for transmission at the different nodes is provided to the M-Server based on message passing, as it would further reduce the already low bandwidth offered by CAN. The Server-CAN framework solves this by always scheduling the N-Servers as if they have messages to transmit, together with polling (monitoring) of the actual traffic on the network.

The Server-CAN framework relies on a single central M-Server, responsible for scheduling the network, and one or more N-Servers that are mediating the access to the network at the different nodes in the system. Moreover, since the centralised approach by using an M-Server is a single point of failure, backup M-Servers can be used. These backup M-Servers are called B-Servers, and take the role of an M-Server in case of M-Server failure (explained in Section 6.3.3). Hence, in total, the system consists of one M-Server, one or more N-Servers, and zero or more B-Servers (depending on the level of fault-tolerance required by the system). These protocol basics are explained in detail below. The system architecture of the Server-CAN framework is depicted in Figure 6.2. Note that the number and allocation of N-Servers, B-Servers and users to the nodes are just examples.
6.3.1 N-Server

Each node connected to the CAN network is assigned one or more network access servers denoted N-Servers, and each N-Server is associated with a share of the system’s total bandwidth. Moreover, each N-Server has one or more associated users. A user is a stream of messages and can be of either periodic or aperiodic nature. The N-Servers mediate access for the users to the local CAN communications adapters.

An N-Server is implemented as a piece of software mediating the access to the network. Actually, the N-Server acts as a software bus guardian (see Section 3.3 for information on bus guardians and babbling idiots), preventing potential damage to the network traffic caused by babbling idiots in the software applications (the users) communicating through the N-Servers. An N-Server has the following responsibilities:

1. To keep its user messages in a local queue. As user messages arrive to the N-Server, they are put in the N-Server’s queue. The local queue is
6.3 Protocol basics

scheduled by some scheduling policy, e.g., FIFO or priority ordered.

2. To receive and read the Trigger Message (TM) sent by the M-Server (more on this in Section 6.3.2). The TM contains the schedule for the upcoming EC, and the N-Server will determine if it is scheduled for message transmission in this EC or not.

3. Every time the N-Server is scheduled for message transmission by the M-Server, the N-Server selects a message (if any) from its local queue and sends it to the local CAN communications adapter.

The N-Server is characterised by a number of parameters and some runtime mechanisms.

N-Server parameters

The following parameters characterise an N-Server:

- **Identity** - each N-Server has a unique integer identity.
- **Capacity** - each N-Server $s$ has an exclusive associated capacity $C_s$.
- **Remaining capacity** - as each N-Server $s$ uses its capacity, it decreases. The remaining capacity $c_s$ is normally (depending on the server policy used) replenished to $C_s$ at the end of a server period. However, using a simple model with $C_s = \text{“1 message”}$, only one message is allowed to be transmitted in each N-Server period.
- **Period** - each N-Server $s$ has an associated period $T_s$ in which the N-Server’s capacity can be used.
- **Bandwidth** - from the capacity and the period, the N-Server’s bandwidth $Q_s$ is defined by $Q_s = C_s/T_s$.
- **Relative deadline** - each N-Server $s$ has a relative deadline $D_s$ usually (depending on the server policy used) set according to the rate monotonic (RM) scheduling policy to $T_s$, i.e., $D_s = T_s$.
- **Absolute deadline** - each N-Server $s$ has an associated absolute deadline $d_s$, that is used by the central scheduler (the M-Server) for scheduling purposes. The initial value of $d_s$ is set to the N-Server relative deadline, i.e., $d_s^0 = D_s$. 

• **User** - each N-Server $s$ has one or more users associated with it. A user $j$ is a stream of messages $m_j$ generated by a subsystem, i.e., an application. This stream of messages can be of either periodic or aperiodic nature.

• **Set of associated messages** - each N-Server $s$ is associated with a set of message identifiers $M_s$ and will only allow transmission of messages with identifiers belonging to this set, i.e., $m_j \in M_s$. This set of messages is used by the M-Server to identify which N-Server a specific message $m_j$ belongs to.

The above mentioned parameters are known by both the N-Server and the M-Server. However, most of them are more important to the M-Server, since they are mainly used to make the scheduling decisions. The parameters are summarised in Table 6.1.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s$</td>
<td>identity</td>
</tr>
<tr>
<td>$C_s$</td>
<td>capacity</td>
</tr>
<tr>
<td>$c_s$</td>
<td>remaining capacity ($c_s \leq C_s$)</td>
</tr>
<tr>
<td>$T_s$</td>
<td>period</td>
</tr>
<tr>
<td>$Q_s$</td>
<td>bandwidth ($Q_s = C_s/T_s$)</td>
</tr>
<tr>
<td>$D_s$</td>
<td>relative deadline ($D_s = T_s$)</td>
</tr>
<tr>
<td>$d_s$</td>
<td>absolute deadline</td>
</tr>
<tr>
<td>$M_s$</td>
<td>the set of associated messages ($m_j \in M_s$)</td>
</tr>
</tbody>
</table>

Table 6.1: N-Server parameters.

### N-Server run-time mechanisms

During run-time, the N-Servers are scheduled by the M-Server according to a share-driven server-based scheduling policy. A server-based scheduling policy is characterised partly by its unique mechanism for assigning and updating absolute deadlines, and partly by a set of parameters used to configure the server. Examples of such parameters are bandwidth, period and capacity. Several server-based scheduling mechanisms are presented in Section 2.3.3.

In order to realise the N-Server run-time functionality, the following mechanisms have to be provided:
6.3 Protocol basics

- **Message queue** - each N-Server has a local queue, in which pending user messages are stored.

- **Scheduler** - each N-Server has a local scheduler that schedules the messages in the local queue. Examples of such schedulers are first in first out (FIFO) queue, or priority ordered queue.

At the N-Server, user messages are queued in a local message queue. This message queue can, for instance, be of either FIFO or priority ordered nature. In the case of a priority ordered message queue the behaviour of the network, from a user point of view, is similar to an exclusive CAN network that has a bandwidth $Q_s$ given by $Q_s = \frac{C_s}{T_s}$, instead of the whole network capacity (which would be $Q_s = 1$).

**N-Server run-time behaviour**

The run-time behaviour of the N-Server is depicted in Figure 6.3. In the figure the thin solid lines represents the state transitions of the state machine, and the thick hollow lines represent how messages are sent from the user(s) to the N-Server’s local message queue to the local CAN communications adapter.

As the N-Server receives a message $M$ from one of its users, it puts this message in its local message queue according to the scheduling policy in use. When the N-Server receives a TM from the local CAN communications adapter, it checks whether or not it is scheduled for message transmission. If the N-Server is scheduled for message transmission it selects a message from the local queue, passing it on to the local CAN communications adapter for message transmission. Upon reception of a message (that is not a TM) from the local CAN communications adapter, if a protocol message $P$ is received, the N-Server performs the Server-CAN request (more on this in Section 6.5). Receptions of all other messages $M$ are passed to the users.

### 6.3.2 M-Server

Standard CAN is strictly priority-based, implementing an FPS scheduler. The Server-CAN protocol is used by the M-Server to schedule CAN according to EDF. The M-Server is a piece of software executing on one of the nodes, the master node. Scheduling CAN using a dedicated “master” is a common way to implement scheduling algorithms on CAN, as presented in Chapter 5. However, the master’s responsibilities differ among these schedulers. For Server-CAN, the M-Server has the following responsibilities:
1. Provide basic run-time mechanisms such as
   
   (a) Schedule the network, i.e., allocate bandwidth to the N-Servers.
   (b) Keep track of the state of each N-Server in the system, i.e., updating the N-Server parameters.
   (c) Reclaim unused bandwidth, i.e., terminate an EC as soon as all scheduled message transmissions are completed.

2. Provide advanced run-time mechanisms such as
   
   (a) Provide admission control.
   (b) Allow for bandwidth sharing.
   (c) Support mode changes.

The M-Server is characterised by a number of parameters, a protocol and some run-time mechanisms.
6.3 Protocol basics

M-Server parameters

The following parameters characterise the M-Server (summarised in Table 6.2):

- **Set of N-Servers** - information on all N-Servers in the system is stored in the M-Server, i.e., all N-Server parameters are stored and maintained by the M-Server. For each N-Server, the parameters given by Table 6.1 are kept.

- **Elementary Cycle** - time is divided into slots called elementary cycles, ECs. This division of time is for scheduling purposes and a schedule of N-Servers allowed to transmit messages (within the EC) is created for each EC. The length of an EC is denoted $T_{EC}$, and kept as a parameter in the M-Server. $T_{EC}$ is the temporal resolution of the CAN network when using Server-CAN, in the sense that N-Servers can not have their N-Server periods shorter than $T_{EC}$.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{N}$</td>
<td>the set of N-Servers in the system</td>
</tr>
<tr>
<td>$</td>
<td>\mathcal{N}</td>
</tr>
<tr>
<td>$T_{EC}$</td>
<td>the length of an EC (in time)</td>
</tr>
<tr>
<td>$</td>
<td>EC</td>
</tr>
</tbody>
</table>

Table 6.2: M-Server parameters.

M-Server protocol

In order for the M-Server to run the Server-CAN protocol, two protocol messages are used:

- **Trigger Message (TM)** - as the M-Server scheduler (described below) has created a schedule for the upcoming EC, this schedule is put (encoded) into a trigger message denoted TM. The TM is then multicasted to all N-Servers in the system. The schedule is encoded as a bit field in the payload data of the TM message. Hence, a total of 64 N-Servers can be scheduled using a single 8 byte CAN message and if the system consists of more than 64 N-Servers, multiple CAN messages must be used for one TM.
Stop message (STOP) - the M-Server uses a stop message, denoted STOP, in order to indicate the termination of an EC. This STOP message has the lowest possible priority, making sure that it will be the last message transmitted when several messages are competing to be transmitted on the network (thanks to the MAC protocol of CAN). Also, the STOP message is of smallest possible size, i.e., 0 bytes of payload data. The usage of a STOP message provides efficient usage of bandwidth, determining when an EC is finished. Note that a small delay is required after the transmission (and reception) of the EC before sending the STOP message since it must be made sure that the STOP message is not sent to the CAN communications adapter before the other nodes have both processed the TM (in order to find out whether they are allowed to transmit messages or not), and (if they are allowed to transmit messages) enqueued their corresponding messages at their corresponding CAN communications adapters.

The reason for using a STOP message to determine the termination of an EC is that it is likely that not all bandwidth allocated within an EC has been completely used. Hence, there is a potential for this unused bandwidth to be reclaimed. There are three sources for unused bandwidth (slack):

1. An N-Server that was allowed to transmit a message (i.e., was scheduled) during the EC did not have any message to transmit.
2. One or more of the transmitted (within the EC) messages were shorter than the assumed worst-case length of a CAN message.
3. The bit-stuffing (see Section 5.5.1) that took place was less than the worst-case scenario.

To not reclaim the unused bandwidth would make the M-Server’s scheduling approach of always schedule N-Servers as if they have messages to send inefficient. Hence, for the proposed method to be viable a mechanism to reclaim bandwidth is needed. In the case that the EC ends early (i.e., due to unused bandwidth) the M-Server reclaims the unused bandwidth by receiving the STOP message and immediately initiating the next EC.

Note that the protocol messages limit the (for users) available bandwidth on the network. The theoretical maximum network utilisation is upper bounded by
6.3 Protocol basics

![Graph showing protocol overhead normalized to the actual length of the EC.](image)

Figure 6.4: Protocol overhead normalised to the actual length of the EC.

\[
\sum_{s=1}^{\lvert N \rvert} \left( \frac{T_M}{T_s} \right) \leq 1 - \left( \frac{T_{TM} + T_{STOP} + T_{sched}}{T_{EC}} \right) \tag{6.2}
\]

where \( \lvert N \rvert \) is the number of N-Servers in the system, \( T_M \) is the length (in time) of a message (typically of worst-case length which is 135 bit-times), \( T_s \) is the period of N-Server \( s \), \( T_{TM} \) is the length (in time) of a TM message (typically 135 bit-times), \( T_{STOP} \) is the length (in time) of a STOP message (typically 55 bit-times), \( T_{sched} \) represents the time required for the M-Server to update the N-Server absolute deadlines, producing the new schedule and encoding it into a TM after receiving the STOP message as well as the decoding of the TM in the slaves, and \( T_{EC} \) is the length (in time) of the EC.

Looking at a specific EC, the protocol overhead is depending on the number of N-Server messages that are transmitted within the EC. This is illustrated in Figure 6.4 where the protocol overhead \( O_i \) is shown normalised to the actual length EC according to

\[
O_i = \frac{T_{TM} + T_{STOP}}{T_{TM} + T_{STOP} + n \times T_M} \tag{6.3}
\]
where \( n \) is the number of N-Server messages transmitted within the EC. What can be seen in Figure 6.4, and of the characteristics of Equation 6.3, is that the protocol overhead is more sensitive when there are few messages transmitted each EC, i.e., the difference in protocol overhead varies more with a lower number of messages transmitted per EC.

**M-Server run-time mechanisms**

During run-time, a number of run-time mechanisms are required to run the M-Server:

- **Scheduler** - all N-Servers are scheduled according to some server-based scheduling policy. The M-Server divides time into ECs and schedules the N-Servers based on their absolute deadlines, creating a schedule for the EC. This schedule is encoded into a TM that is multicasted to all N-Servers in the system, indicating the start of an EC.

  When the M-Server is performing the scheduling, all N-Servers are scheduled as if they have messages to transmit, and only eligible N-Servers are considered.

**Definition 6.3.1.** An arbitrary N-Server \( s \) is eligible at time \( t \) iff the following inequality holds

\[
d_s - t \leq D_s
\]  

(6.4)

The reason for introducing the eligible constraint is that when transmitting messages, N-Servers are to be scheduled consuming their capacity once every N-Server period. This will guarantee their temporal performance as presented in the remainder of this thesis. One could consider to also schedule non-eligible N-Servers when there are not enough eligible N-Servers to fill one schedule. This is briefly discussed in Chapter 10. However, in this thesis, only eligible N-Servers are scheduled by the M-Server.

When scheduling a new EC, the M-Server will (in EDF order) select the eligible N-Servers that are allowed to transmit messages in the EC, creating a schedule. Next, the M-Server encodes this schedule into a TM and multicasts (transmits) the TM to all N-Servers. The reception of a TM at an N-Server indicates that a new EC has been initiated by the M-Server, and the TM contains information on which N-Servers that are allowed to transmit messages during the EC, i.e., the EC’s schedule.
Upon reception of a TM, the N-Servers allowed to transmit messages (the scheduled N-Servers) will enqueue their messages in their corresponding CAN communications adapters. The messages enqueued by the scheduled N-Servers will then, in turn, be transmitted using the standard CAN MAC protocol. Due to the arbitration mechanism of CAN, it is hard to know when inside an EC a specific message is transmitted. Hence, due to this uncertainty of not knowing when a message is transmitted within an EC, the bound on the delay of message transmissions is affected by the size of the EC (more on this in the upcoming two chapters).

Once the TM has been transmitted, the M-Server has to determine when all scheduled N-Servers have transmitted their messages, so the start of a new (the next) EC can be initiated. The M-Server does this by transmitting the STOP message with the lowest possible priority.

• **Run-time monitoring** - the M-Server continuously polls (monitors) the traffic on the CAN network, updating the N-Server parameters according to the actual traffic on the network. By doing this, the M-Server is able to verify whether the scheduled N-Servers had messages to send or not, and update the respective N-Server’s parameters accordingly. Depending on how these parameters are updated, different temporal performance can be achieved, as shown in Chapter 8 for PS²-CAN and S³-CAN.

After sending the STOP message to the local CAN communications adapter, the M-Server monitors all messages transmitted on the network during the EC. When the M-Server receives the STOP message it knows that all N-Servers that were scheduled to send messages in the EC have transmitted their messages (if they had any messages to send). After the reception of the STOP message the EC is over and the M-Server updates (if not already updated) the parameters of the scheduled (during the EC) N-Servers, before initiating the next EC.

Another way of determining when the EC is finished is that the CAN communications adapter itself is able to determine when the network becomes idle. If this is possible, there is no need for the STOP message. However, by using a STOP message it is possible to use standard CAN communications adapters.

Also, timers can be used to monitor the time since the last message was received at (polled by) the M-Server and based on this information the
EC can be terminated. Using this approach, standard CAN communications adapters can be used.

- **Support for advanced run-time mechanisms** - the M-Server implements admission control, bandwidth sharing mechanisms and support for mode changes. These mechanisms are controlled using specific protocol messages, further explained in Section 6.5.

### M-Server run-time behaviour

The run-time behaviour of the M-Server is depicted in Figure 6.5. In the figure the thin solid lines represent the state transitions of the state machine, the thick hollow lines represent how messages are sent between the M-Server and the local CAN communications adapter, and the dashed lines represent updating and extraction of parameters.

As the M-Server receives a STOP message, the schedule for a new EC is produced and encoded in a TM. This TM is multicasted to all N-Servers as the initiation of a new EC. When the M-Server receives a TM message from the communications adapter it waits for some time before sending a STOP message. The waiting is to make sure that the TM is decoded at all the N-Servers in the system and that they (if scheduled) have enqueued their messages in their corresponding CAN communications adapters. Also, the M-Server can receive a protocol specific message P, used to control the advanced run-time mechanisms such as admission control, mode changes and bandwidth sharing. Upon reception of these messages, appropriate mechanisms are initiated. Finally, reception of all other messages M triggers the updating of the N-Server parameters.

### 6.3.3 B-Servers

To provide fault-tolerance, the M-Server is replicated using backup M-Servers called B-Servers. These B-Servers need to be consistent, and how to maintain this consistency is an issue of ongoing work, not yet implemented. Below, the main ideas of the B-Server concept are outlined.

**B-Server characterisation**

The B-Server is characterised by the same parameters as the M-Server. In fact, a B-Server is an M-Server with an extra state. The two states of a B-Server
are “B-Server state” and “M-Server state”. As long as the B-Server stays in the B-Server state, it will not send any protocol messages (TM and STOP messages as well as messages controlling the advanced run-time mechanisms presented in Section 6.5), but it will do all other things that an M-Server do. However, in case of an M-Server failure one B-Server will take the full role as the new M-Server, i.e., the B-Server enters the M-Server state.

**B-Server run-time behaviour**

The run-time behaviour of the B-Server is depicted in Figure 6.6. In the figure the thin solid lines represents the state transitions of the state machine, the thick hollow lines represent how messages are sent between the M-Server and the local CAN communications adapter, and the dashed lines represent updating and extraction of parameters.

The difference between the M-Server and the B-Server, is that the B-Server receives and use the TM sent by the M-Server to check that it is consistent with the TM that it would produce itself (if it were to be the M-Server), i.e., a
consistency check. Also, upon reception of a STOP message, a B-Server does not multicast a TM. Instead, the newly created TM is stored in the B-Server for a later consistency check with the received TM (sent by the M-Server). If the TM of the B-Server is not consistent with the contents of the TM sent by the M-Server, a synchronization process must be initiated in order to make the M- and B-Servers consistent.

### 6.4 Example basic run-time scenario

The basic run-time mechanisms, such as, scheduling of the N-Servers, updating of the N-Server parameters and reclaiming of the unused bandwidth are below explained with a run-time scenario. This scenario is depicted in Figure 6.7, and the state machines (indicated in the figure) for the N-Server and M-Server are given in Figure 6.3 and Figure 6.5 respectively.

The scheduling of the N-Servers is performed at the M-Server according to EDF (Figure 6.7:1, i.e., (1) in Figure 6.7). A schedule is created containing
the 4 N-Servers with the earliest absolute deadlines, filling up one EC (in this example, an EC fits 4 messages). As this is done, the schedule is encoded into a TM and multicasted to all the N-Servers in the system (Figure 6.7.2). In the example, the schedule (the TM) is containing “message 1”, “message 4”, “message 5” etc. which is shown as “M1”, “M4” and “M5” in the figure.

When the N-Servers receive the TM (Figure 6.7.3), they will decode it to see whether they are allowed to send a message or not (Figure 6.7.4). If they are allowed to send a message, their message is (if existing) immediately queued for transmission at the local CAN communications adapter (Figure 6.7.5). Otherwise, if not scheduled, the N-Server has to wait for the next TM to see if it was scheduled that time.

Also, the M-Server receive the TM (Figure 6.7.6) and waits for a while
before sending a STOP message to itself. The STOP message is of lowest priority possible, acting as an indicator for when all the N-Servers that were scheduled for message transmission within the active EC actually have transmitted their messages. If the M-Server receives the STOP message, all N-Servers that were allowed to send messages within the EC have already transmitted their messages. This since the STOP message, with its low priority, is the last message to be transmitted within the EC.

The M-Server is always reading (receiving) all the messages that are sent (transmitted) on the CAN network, i.e., the M-Server is polling the bus. This in order to update all N-Server parameters based on the actual traffic on the network (i.e., based on the transmitted messages). Since N-Servers are scheduled for message transmission even though they might not always have any messages to send, this has to be taken care of by the M-Server, by updating the N-Server parameters accordingly. There are different ways of updating the N-Server parameters in the case when the N-Server did not transmit a message. Depending on how the N-Server parameters are updated the N-Server will have different real-time characteristics, as shown in Chapter 7.

When the M-Server receives the STOP message, the EC is terminated, and the next EC is initiated based on the updated N-Server parameters.

### 6.5 Advanced run-time mechanisms

Advanced run-time mechanisms can be used to provide admission control, i.e., allowing dynamic adding and removing of N-Servers in the system. Using the Server-CAN concept, N-Servers and users can potentially join and leave the system arbitrary as long as the total utilisation (bandwidth demand), by all the N-Servers is less or equal to the theoretical maximum given by Equation 6.2. Note that the joining and leaving of users does not affect other users (in terms of temporal guarantees) than those sharing the same N-Server. This is due to the bandwidth isolation between different N-Servers. Also, providing support for mode changes allow the system to change from one operational mode to another. Finally, to further increase the dynamic flexibility of the Server-CAN framework, providing mechanisms for bandwidth sharing allow the system to evolve during run-time utilising the bandwidth resources more efficiently.
6.5 Advanced run-time mechanisms

6.5.1 Admission protocol

As an option to add flexibility to the Server-CAN framework, an admission control mechanism can be used to dynamically add and remove N-Servers at run-time. In order for the admission control to work, each node in the system is required to have an N-Server that can be used to transmit protocol messages, e.g., an N-Server that is used for non real-time traffic.

Each N-Server has an N-Server ID \( s \), and is allowed to send messages with specified identifiers \( m_j \). Hence, each N-Server \( s \) is associated with a set of message identifiers \( M_s \) and will only allow message transmission of messages with identifiers \( m_j \in M_s \).

Each node \( n \) is allocated a specific message identifier \( m_n \) used for protocol specific messages. These message identifiers are known and allocated to nodes at the initiation of the system. Hence, the M-Server will read these messages as protocol messages, automatically decoding the message format described below. All protocol message identifiers \( m_n \) are kept at the M-Server in a set of protocol message identifiers \( P \).

Implementation

When implementing the admission control, three types of requests are encoded into a single request-message with the following format:

\[
\langle \text{updateCMD}, \text{nodeID}, \text{serverID}, \text{C}, \text{T}, \text{m} \rangle
\]

These request messages are used to add an N-Server, remove an N-Server, and update N-Server parameters. Moreover, their corresponding reply-messages all have the following format:

\[
\langle \text{replyCMD}, \text{nodeID}, \text{serverID}, \text{value} \rangle
\]

Hence, two types of messages are involved in the admission control mechanism, namely request- and reply-messages.

Adding of N-Server

Here a request-message is sent containing \( \text{updateCMD} = \text{addNServer} \) and \( \text{nodeID} \) = the N-Server ID of the requesting N-Server \( s \). \( \text{C} \) and \( \text{T} \) are the server-parameters \( C_j \) and \( T_j \) of the requested new N-Server \( j \), and \( m \) is its associated message identifier \( m_j \). Note that \( m_j \) is provided in the request to allow for adding of systems with already predetermined message identifiers. Also, if the N-Server \( j \) is to be associated with multiple message identifiers, sev-
eral request-messages have to be sent, using $\text{updateCMD} = \text{updateNServer}$ as described below.

When the M-Server receive this message, a schedulability test is performed (the admission control) and a reply-message is sent containing $\text{replyCMD} = \{\text{admitted, rejected}\}$ and $\text{nodeID} =$ the N-Server ID of the requesting N-Server $s$. If admitted, $\text{serverID} =$ N-Server ID of the new N-Server $j$, allocated by the admission control. Moreover, the M-Server add $m_j$ to $\mathcal{M}_j$, i.e., the requested message identifier is associated with the new N-Server $j$.

When the requesting N-Server $s$ receive the reply-message a new N-Server $j$ is spawned with N-Server ID set to $\text{serverID}$, only allowing message transmissions with message identifier $m_j$.

Removing of N-Server

Here a request-message is sent containing $\text{updateCMD} = \text{removeNServer}$ and $\text{nodeID} =$ the N-Server ID of the requesting N-Server $s$. $\text{serverID} =$ the N-Server ID of the N-Server $j$ that is to be removed (possibly $s = j$).

When the M-Server receives this message, N-Server $j$ is removed from the M-Server parameters, and a reply-message is sent with $\text{replyCMD} = \{\text{removed, fail}\}$ and $\text{nodeID} =$ the N-Server ID of the requesting N-Server $s$. Also, $\text{serverID} =$ the N-Server ID of the removed N-Server $j$.

When the N-Server $s$ receives this message the N-Server $j$ closes down.

Updating of N-Server

Here a request-message is sent containing $\text{updateCMD} = \text{updateNServer}$ and $\text{nodeID} =$ the N-Server ID of the requesting N-Server $s$. $\text{serverID} =$ the N-Server ID of the N-Server $j$ that is to be updated and $C$ and $T$ are the proposed new N-Server parameters $C_j$ and $T_j$, and $m$ is a new message identifier $m_j$ to be associated with the N-Server $j$.

By setting both $C_j$ and $T_j$ to values larger than 0 indicates an update of these parameters. By setting $C_j = 0$ indicates an adding of a message $m_j$ to an existing N-Server $j$, and by setting $T_j = 0$ indicates the removal of a message $m_j$ from an existing N-Server $j$.

When the M-Server receives this message, a schedulability test is performed (the admission control) and a reply-message is sent where $\text{replyCMD} = \{\text{changed, fail}\}$ and $\text{nodeID} =$ the N-Server ID of the requesting N-Server $s$. If admitted, the M-Server updates its parameters. Moreover, if requested, the M-Server add $m_j$ to $\mathcal{M}_j$. 
6.6 Fault tolerance

6.5.2 Mode changes

When the system performs a mode change, also the M-Server has to change its server parameters from one set to another. The mode change is triggered by the reception of a mode-change message with the following format:

\(<\text{modeChangeREQ}, \text{nodeID}, \text{mode}\>

Upon the reception of this message the M-Server changes to mode \(\text{mode}\), as indicated in the message. The \(\text{nodeID}\) is the N-Server ID of the requesting N-Server.

6.5.3 Bandwidth sharing

The Server-CAN framework allows for the implementation of bandwidth sharing mechanisms. By having the M-Server monitoring the traffic it is possible to share bandwidth between N-Servers, even changing their N-Server parameters. Bandwidth sharing is done directly in the M-Server using bandwidth sharing algorithms, e.g., [18, 120], and the changing of N-Server parameters (\(C_j\) and \(T_j\)) is done using the admission control presented above.

6.6 Fault tolerance

The system may suffer from different types of faults, namely, node software faults, node physical faults, channel permanent faults and channel transient faults. The impact of these faults on the Server-CAN framework have been investigated [153], however this work is not yet completed. In [153], inherent mechanisms of Server-CAN in order to achieve fault-tolerance, as well as mechanisms which could be easily incorporated, are discussed.

Message omissions are recovered by retransmissions, provided by the built-in fault tolerance ability of the standard CAN protocol. Server-CAN builds on several mechanisms (both inherent and supplemented) in order to facilitate a fault-tolerant operation.

6.6.1 Fault model

The fault model used is that the network can suffer from four types of faults: (1) node software faults, (2) node physical faults, (3) channel permanent faults, and (4) channel transient faults. Each of these faults is discussed together with its
Impact on the communications when using Server-CAN, and how the Server-CAN protocol is affected. In some cases, feasible solutions for overcoming these faults are also discussed.

**Node software faults**

By using N-Servers as an application’s interface to the network, bad and unexpected behaviour of one application will not propagate to other applications on other nodes. Applications on the same node might suffer, and there might be a possible overload in the badly behaving application’s N-Server. Also, propagation of errors from the badly behaving application could corrupt the N-Server.

N-Servers are implementing a software bus guardian preventing babbling idiots. Solutions to the babbling idiot problem have been presented for CAN [22] by the usage of extra hardware. Server-CAN is a software solution to this problem. However, compared with hardware bus-guardians, Server-CAN only solves babbling idiots caused by software faults. Also, Server-CAN does not present fault independence with respect to the rest of the node and is therefore vulnerable to propagation of errors from applications running on the same node.

Node software faults typically stems from software design errors. Here, it is assumed that all software, including the M-Server and the N-Server, are properly designed and subjected to validation tests and if possible, formally verified. The M-Server and the N-Server are not complex and therefore unlikely to exhibit design faults. However, other applications might suffer from software faults and then it is important to determine how these faults affect the Server-CAN communication.

**Node physical faults**

The M-Server is a single point of failure in the system, and the probability of a physical fault in the M-Server is not negligible. If the M-Server crashes there will be no transmission of the TM, i.e., no schedule is sent to the N-Servers in the systems and the system could in the worst-case be blocked. For correct operation, the M-Server must always recover. A solution to this is to have replicated M-Servers to prevent the scheduler of the network to disappear. This replication of the M-Server can be done similar to the handling of master-nodes in FTT-CAN [49, 177].

The replication of the M-Server is achieved by having one or more backup M-Servers (called B-Servers) in the system. These hot standby B-Servers keep
monitoring the network as normal M-Servers, updating its server states, but they do not send TMs. Moreover, as N-Servers join or leave or change their properties, the B-Servers update their information as well. The consistency of this information must be guaranteed, and solutions similar to those existing for FTT-CAN [177] could be used. All B-Servers can verify that the TM contains the correct information. If not, a synchronisation of M-Server data can be done. When there is no transmission of a TM by the M-Server, a B-Server takes on the role as the M-Server and transmits its current TM.

From a system point of view, the N-Server is not a single point of failure. However, it is a single point of failure from its users’ point of view.

Channel permanent faults

Channel permanent faults include link partitions, stuck-at-dominant, etc. The single link of a bus topology is a single point of failure. In topologies different from a bus, e.g., a star, a faulty link does not cause a global failure of the system. These types of faults are very important and usually addressed by bus replication [179]. Hence, in this thesis it is assumed that the channel is free from permanent faults or is able to tolerate them by its own means.

Channel transient faults

These faults are due to Electro Magnetic Interference (EMI) and cause message duplications and message omissions, and can be either consistent or inconsistent [180]. Message duplications cause a message to be transmitted twice at the cost of loss of bandwidth, and message omissions cause a message not to be transmitted at all.

Using the Server-CAN protocol, the M-Server is responsible for scheduling all N-Servers and sending the schedule to the N-Servers using the TM and terminating the EC using a STOP message. Hence, protocol specific messages sensitive to channel transient faults are the TM and the STOP messages, for which the implications of a channel fault during their transmission have to be investigated. Also, since these faults can happen in different combinations, detailed analysis of their impact on the fault-tolerant operation of CAN and Server-CAN is essential. Inconsistent message omissions may jeopardize the consistency among the M-server replicas.
6.6.2 Consistency of M- and B-Servers

Looking at the fault model in the context of Server-CAN, the main issue is concerning consistency among the M- and B-Servers, i.e., how to cope with channel transient faults. As the M- and B-Servers are continuously monitoring the traffic on the network in order to update their N-Server parameters, inconsistent message omissions and duplications could cause inconsistencies, i.e., the N-Server parameters are not consistent among the M- and B-Servers. This has been solved for FTT-CAN in [177], however, the situation for Server-CAN is slightly more complicated as here all messages are potential sources for inconsistencies (not only the protocol messages as is the case in [177]). This work on consistencies among replicated M- and B-Servers is not finished yet, but part of the future work discussed in Chapter 10.

Also, due to the complexity introduced by the advanced protocol mechanisms of the Server-CAN framework, the following mechanisms are subject to inconsistency among the M and B-Servers and have to be analysed in detail:

- **Admission control** - The admission protocol involves message passing. Hence, it is vulnerable to channel transient faults. The admission protocol involves the transmission of request- and reply-messages. To ensure consistency, similar techniques as for FTT-CAN can be used [177].

- **Bandwidth sharing** - As bandwidth sharing mechanisms involve changing M-Server parameters, special consideration needs to be taken in order to avoid inconsistencies among the replicated M-Servers. Here, the same messages as when updating N-Server parameters are used. Hence, bandwidth sharing suffers from the same fault scenarios as admission control.

- **Mode changes** - Changing mode involves message passing and is therefore vulnerable to channel transient faults.

6.7 Summary

This chapter presents Server-CAN, a new share-driven scheduling framework for CAN. In the chapter, scheduler design decisions are discussed and basic run-time mechanisms are presented together with the scheduling protocol. Also, several advanced run-time mechanisms allowing for admission control, mode changes and bandwidth sharing are described.
In order to use Server-CAN in safety-critical applications its fault tolerant operation has also to be analysed. At the end of this chapter, the fault model intended to be used for the fault-tolerant operation of Server-CAN is presented. However, only initial results are presented as this is part of ongoing research.

In the next chapter, two different Server-CAN schedulers are presented and their worst-case response-time when transmitting a single message is analytically determined and proven. Also, one reference scheduler is presented, intended to be used for evaluation purposes in Chapter 8.

Note that the focus of the remainder of this thesis is on how to realise different Server-CAN schedulers and investigating the real-time performance that can be guaranteed for these schedulers. This since the topic of the thesis is on share-driven scheduling for embedded networks, not on flexible performance (or fault-tolerant operation). The scheduler is used to facilitate subsystem integration from a network point of view. Hence, in this context, the Server-CAN advanced run-time mechanisms and the fault-tolerant operation of Server-CAN are not as interesting (from the thesis point of view) as Server-CAN’s real-time properties, and are therefore not fully evaluated in the thesis but left for future work.
Chapter 7

Server-CAN schedulers

The previous chapter presented the basic properties and mechanisms of Server-CAN, a share driven scheduler running on top of standard CAN. This chapter presents and evaluates two Server-CAN schedulers called Periodic Server-Scheduled CAN (PS\(^2\)-CAN) and Simple Server-Scheduled CAN (S\(^3\)-CAN). The schedulers differ in the way the M-Server updates the N-Servers’ absolute deadlines in the case when a scheduled N-Server did not transmit a message. As shown in the chapter, the deadline updating strategy affects the timing of transmitted messages. The main contributions of this chapter can be summarised as:

- Periodic Server-Scheduled CAN (PS\(^2\)-CAN) is presented. PS\(^2\)-CAN is a simple Server-CAN scheduler, implementing a dynamic priority bandwidth conserving variant of the Polling Server (PS) [111, 190, 195].

- Simple Server-Scheduled CAN (S\(^3\)-CAN) is presented. S\(^3\)-CAN is a Server-CAN scheduler implementing a bandwidth conserving polling version of the Total Bandwidth Server (TBS) [197, 199].

- Worst-case response-time analysis is presented for both PS\(^2\)-CAN and S\(^3\)-CAN, bounding the maximum theoretical delay for an arbitrary N-Server to transmit a message.

- A scheduler that is to be used for comparison is presented as well, together with its associated worst-case response-time analysis. This scheduler, called Periodic Polling scheduling mechanism (PP-CAN), is used in the evaluations performed in Chapter 8.
First, general Server-CAN properties introduced by the Server-CAN protocol messages (TM and STOP) are investigated. Following this, PS\(^2\)-CAN and S\(^3\)-CAN are presented together with their associated timing analysis. Finally, PP-CAN is presented, also with its associated timing analysis.

### 7.1 General Server-CAN properties

Some properties of Server-CAN schedulers are valid for all Server-CAN schedulers, no matter how they are implemented. A general property of the Server-CAN scheduling framework is that the protocol overhead increases when the network utilisation is low. When the network is not fully utilised, the M-Server will transmit more TM and STOP messages than it would during full network utilisation. Also, the protocol overhead increases when scheduled N-Servers have no messages to transmit. This is illustrated in Figure 7.1, where Figure 7.1a shows a fully utilised network where all N-Servers that are allowed to transmit messages do so. Figure 7.1b shows what happens if not all N-Servers transmit messages, causing TM and STOP messages to be transmitted more frequently. In fact, using the Server-CAN scheduling framework, the protocol messages will always consume all “free” bandwidth of the network, regardless of the number of N-Server messages actually ready for transmission (and being transmitted). Hence, there are always transmissions of messages; TM, STOP and scheduled N-Server messages.

This increase of protocol overhead, when N-Servers do not transmit their M-Server scheduled messages, makes determining the temporal behaviour of some Server-CAN scheduling mechanisms (e.g., the S\(^3\)-CAN scheduler) more complicated than response-time analysis of “traditional” server-based schedulers for CPU scheduling.

What characterise a Server-CAN scheduler is the way the M-Server updates its N-Server parameters. Different rules for updating the N-Server parameters are enforced by different Server-CAN schedulers. The updating of N-Server parameters can be done directly after the M-Server monitors a specific message belonging to an N-Server, or at the termination of the EC, i.e., after reception of the STOP message. This is depicted in Figure 7.2.

When using Server-CAN, time is divided into ECs no matter which Server-CAN scheduler that is used. Due to this division of time into ECs, together with the decoupling of message priorities from message periods, an arbitrary message might be sent at an arbitrary time within an EC.
Assumption 7.1.1 (Fault free message transmissions). In this chapter it is assumed that CAN provides fault free message transmissions, i.e., message omissions and duplications are not taken into account. It is assumed that all transmitted messages are successfully transmitted.

Lemma 7.1.2. When an arbitrary N-Server \( s \) is scheduled for message transmission in an EC, its message will in the worst-case be transmitted as the last message within the EC, regardless of the exact value of its absolute deadline \( d_s \).

Proof. The N-Server absolute deadlines are used by the M-Server to decide which N-Servers to schedule, as the M-Server scheduler implements EDF. Once an N-Server is scheduled for message transmission within an EC, it will directly send a message to its CAN communications adapter. However, as messages are sent to the CAN communications adapters by all N-servers that are scheduled for message transmission within the EC, a number of messages (maximum \(|EC|\) messages) will be scheduled according to the CAN message arbitration mechanism, i.e., the messages will be scheduled based on their priorities. As the priority of a message is (at the communications adapter) represented (given) by its identifier, and as the message identifier can be arbitrary...
All N-Server absolute deadlines are updated according to actual traffic, done either here or directly after a N-Server specific message transmission. New schedule created by the M-Server, encoded into a trigger message (TM), and multicasted to all N-Servers.

Figure 7.2: Updating of N-Server parameters.

for an arbitrary message sent through an arbitrary N-Server, a message might be transmitted at any time within the EC (since it all depends on the messages competing to be transmitted within the same EC). The only thing guaranteed is that an N-Server scheduled for message transmission within an EC will have its message transmitted no later than before the termination of the EC, i.e., in the worst-case an arbitrary message will be transmitted as the last message within the EC.

**Definition 7.1.3** (System load). The total system load, denoted $Q_{system load}$, is defined as the load imposed by the system's set of N-Servers (denoted $N$) on the system. $Q_{system load}$ is upper bounded by

$$Q_{system load} = \sum_{s=1}^{\lvert N \rvert} \left( \frac{T_M}{T_s} \right)$$

where $\lvert N \rvert$ is the number of N-Servers in the system, $T_M$ is the length (in time) of a message (typically of worst-case length which is 135 bit times), $T_s$ is the period of N-Server $s$.  


7.2 Periodic Server-Scheduled CAN (PS²-CAN)

Definition 7.1.4 (Feasibility). The system is said to be feasible if it is theoretically possible to run the system with the whole set of N-Servers $N$ and also calculate a predictable worst-case response-time.

In order for the system (consisting of $|N|$ N-Servers) to be feasible, the following inequality must hold, i.e., the system load including protocol overhead, computational overhead, and the transmission of messages when all scheduled N-Servers transmit their corresponding messages, must be less than 100%:

$$Q_{\text{system\,load}} \leq 1 - \left( \frac{T_{TM} + T_{STOP} + T_{sched}}{T_{EC}} \right)$$

(7.2)

where $T_{TM}$ is the length (in time) of a TM message (typically 135 bit times), $T_{STOP}$ is the length (in time) of a STOP message (typically 55 bit times), $T_{sched}$ represents the time required for the M-Server to update the N-Server absolute deadlines, producing the new schedule and encoding it into a TM after receiving the STOP message as well as the decoding of the TM in the slaves, and $T_{EC}$ is the length (in time) of the EC.

7.2 Periodic Server-Scheduled CAN (PS²-CAN)

In this section the Periodic Server-Scheduled CAN (PS²-CAN) is presented. PS²-CAN is a dynamic priority bandwidth conserving variant of the Polling Server (PS) [111, 190, 195]. The worst-case response-time for an arbitrary N-Server $s$ is shown for PS²-CAN to be mainly dependent on the N-Server relative deadline $D_s$ (which for PS²-CAN is equal to the N-Server period $T_s$).

In general, scheduling an N-Server that has no messages ready for transmission causes a loss of bandwidth. PS²-CAN takes this lost bandwidth from the N-Server that had no messages ready for transmission when scheduled, whereas S³-CAN (presented in Section 7.3) takes the lost bandwidth from the system (i.e., all N-Servers in the system have to share the loss of bandwidth). The different strategies give different temporal performance, as is shown in the analysis of the worst-case response-time for PS²-CAN and S³-CAN.

7.2.1 Updating N-Server states

Using PS²-CAN the M-Server always treat an N-Server that has been scheduled for message transmission as if the N-Server actually did transmit a message (regardless whether it did transmit any message or not). Hence there is only a single rule for updating the N-Server parameters:
• Treat the N-Server $s$ as if it transmitted a message, and update its N-Server absolute deadline $d_s$ according to

$$d^n_s = d^{n-1}_s + D_s$$  \hspace{1cm} (7.3)

where $D_s$ is the N-Server relative deadline. This deadline updating scenario is depicted in Figure 7.3, where the N-Server absolute deadline $d_s$ is updated by the M-Server at time $t$, after termination of the EC where the N-Server was scheduled for message transmission.

![Figure 7.3: Updating of N-Server absolute deadline for PS $^2$-CAN.](image)

### 7.2.2 M-Server scheduling

In order to prevent an N-Server that has no messages ready for transmission to be scheduled more than once during a time-span of its N-Server period, the M-Server apply the following rules when creating the schedule for the EC starting at time $t$

1. Only N-Servers that are within one N-Server period $T_s$ from their absolute deadline $d_s$ are eligible for scheduling, i.e., the time remaining before the absolute deadline (the laxity) must be less or equal to the N-Server’s relative deadline $D_s$. Hence, the following condition must hold for an N-Server $s$ to be eligible for scheduling:

$$d_s - t \leq D_s$$  \hspace{1cm} (7.4)
2. Among the eligible N-Servers, select N-Servers in EDF order to be scheduled during the EC.

Consider the example depicted in Figure 7.4. Here, the M-Server will at time \( t \) select eligible N-Servers fulfilling Inequality 7.4 in the following order: N-Server 5, N-Server 1, N-Server 6 and N-Server 4. The non-eligible N-Servers will not be scheduled.

As a motivating example to illustrate the impact of scheduling an N-Server with no messages ready for transmission (when using PS\(^2\)-CAN), consider a system of 3 N-Servers (\(|\mathcal{N}| = 3\) having N-Server periods \( T_1 = 6, T_2 = 12, T_3 = 12 \). Each N-Server is in each N-Server period allowed to send one (1) message with a constant transmission time (i.e., \( C_i = 1, i \in \{1, 2, 3\} \)). Each EC contains, for simplicity, only one N-Server message, i.e., \(|EC| = 1\). The length of the EC is set to 3, \( T_{EC} = 3 \), hence the system is fully utilised (i.e., system load = 100\%). The system parameters are summarised in Table 7.1.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(</td>
<td>\mathcal{N}</td>
</tr>
<tr>
<td>(T_{EC})</td>
<td>3</td>
</tr>
<tr>
<td>(</td>
<td>EC</td>
</tr>
<tr>
<td>(C_i, i \in {1, 2, 3})</td>
<td>1</td>
</tr>
<tr>
<td>(T_i)</td>
<td>6</td>
</tr>
<tr>
<td>(T_2, T_3)</td>
<td>12</td>
</tr>
<tr>
<td>(D_i, i \in {1, 2, 3})</td>
<td>(T_i)</td>
</tr>
</tbody>
</table>

Table 7.1: Small example system.
7.2.3 Timing analysis

In order to provide a worst-case response-time analysis for PS$^2$-CAN some of its basic properties are investigated. The worst-case response-time is derived from the fact that the scheduling algorithm of PS$^2$-CAN will behave no worse than if EDF scheduling was used to schedule messages (all messages including the TM and STOP messages).
Lemma 7.2.1. During a time interval, if all N-Servers that are scheduled for message transmissions do transmit their messages, the scheduling mechanism will emulate pure EDF. Hence, during a time interval, all N-Servers will be scheduled for message transmission in an EC once every N-Server period if the total system load is less than 100%.

Proof. As long as the system load is less than 100%, as the N-Servers are scheduled according to EDF, when all scheduled N-Servers transmit their messages they will behave like periodic processes according to the PS$^2$-CAN rule of updating the N-Server’s absolute deadline (Equation 7.3). What is required for the N-Servers to be scheduled once every N-Server period is that the system is feasible, i.e., that the total system load is less than 100% as expressed by Inequality 7.2.

Lemma 7.2.2. If an N-Server that is scheduled for message transmission does not transmit a message, no other N-Server will miss its deadline because of this.

Proof. When an N-Server scheduled by the M-Server does not transmit a message, the EC will be terminated early (as can also happen if not all messages are of maximum length). This is depicted in Figure 7.6, where the EC is terminated at time $t_b$ (actual schedule) instead of at time $t_a$ (intended schedule).

![Figure 7.6: Early EC termination on PS$^2$-CAN.](image)

However, since the N-Servers that do not transmit messages when scheduled will not be scheduled again until within their next N-Server period they cannot cause increased protocol overhead in terms of TM and STOP messages being generated (which would have been the case for S$^3$-CAN presented in...
Section 7.3). This is true since the transmission of the (one) early STOP message could be thought of as replacing the lacking N-Server message, as it is of smaller size compared to the N-Server’s associated (not transmitted) message (the STOP message is of smallest size possible).

Also, since the current EC is terminated early, at time $t_b$ instead of time $t_a$, subsequent messages from other N-Servers that are to be scheduled in forthcoming ECs will be served earlier than they would have been if the N-Server did transmit its message (and, hence, they cannot miss their deadlines). If the schedule following after time $t_a$ is feasible, the same schedule is feasible at time $t_b$.

**Theorem 7.2.3.** Using PS$^2$-CAN, the worst-case response-time $R_s$ for the transmission of a message by an N-Server $s$, given that N-Server $s$ has a message to transmit, is given by

$$R_s = 2 \times T_s + T_{EC} - T_{STOP}$$  \hspace{1cm} (7.5)

**Proof.** Lemmas 7.2.1 and 7.2.2 tells that N-Server $s$ will be scheduled at least once during its N-Server period $T_s$. There is however no restriction on when, during $T_s$, the N-Server is scheduled. In the worst-case, the distance between two consecutive occasions of scheduling N-Server $s$ then becomes $\sim 2 \times T_s$. This worst-case distance between two scheduled instances of an N-Server occurs if the first time it is scheduled immediately in its $T_s$ and the second time it is scheduled as late as possible in its $T_s$.

However, it is only known that the second time the N-Server will be scheduled for message transmission is no later than the EC in which its absolute deadline $d_s$ is located. Due to the phases between the M-Server, the EC and the N-Servers, it is unknown when during this EC the N-Server absolute deadline occurs. As the order of message transmissions within this EC is unknown (Lemma 7.1.2), it must be pessimistically assumed that the N-Server’s message is transmitted as the last message in the EC (and that the N-Server’s absolute deadline is at the start of the EC). This is shown in Figure 7.7, and will incur an extra maximum delay of $T_{EC} - T_{STOP}$ for the message.

### 7.3 Simple Server-Scheduled CAN (S$^3$-CAN)

Simple Server-Scheduled CAN (S$^3$-CAN) is a Server-CAN scheduler that implements a bandwidth conserving polling version of TBS [197, 199], in the
sense that the deadline updating strategy is similar to the one of TBS and the capacity of each N-Server is set to one (1) message. Hence, an arbitrary N-Server is at most allowed to transmit one message each EC. Below, $S^3\text{-CAN}$ is presented in detail together with its associated timing analysis.

### 7.3.1 Updating N-Server states

Looking at $S^3\text{-CAN}$, as the M-Server receives the STOP message, the EC is terminated and the absolute deadlines of all (within the EC) scheduled N-Servers are (or have been during the polling) updated according to whether they transmitted messages or not. For $S^3\text{-CAN}$ the following rules apply for updating the N-Server absolute deadlines:

1. If the scheduled N-Server transmitted a message, the M-Server updates the N-Server’s absolute deadline $d_s$ according to

   \[
   d_s^n = d_s^{n-1} + D_s
   \]

   where $D_s$ is the N-Server relative deadline. This scenario is depicted in Figure 7.8a.

2. If the scheduled N-Server did not transmit a message, there were no messages ready for transmission at the N-Server, and the N-Server’s absolute deadline $d_s$ is updated according to
Figure 7.8: Updating of N-Server absolute deadline for $S^3$-CAN.

\[ d^n_s = \max(t + D_s, d^{n-1}_s) \]  \hspace{1cm} (7.7)

where $t$ is the time of the scheduling, and $D_s$ is the N-Server relative deadline. This deadline updating scenario is depicted in Figure 7.8b.

Looking at the above rules for updating the N-Server absolute deadlines, they are similar to a TBS server with capacity set to one (1) message. However, scheduling N-Servers with no messages to transmit consumes bandwidth, and therefore $S^3$-CAN can not be as efficient as the original TBS scheduler.

### 7.3.2 M-Server scheduling

In order to prevent an N-Server with no messages ready for transmission to be scheduled multiple times during a time-span of its N-Server period, the M-Server apply the following rules when creating the schedule for the EC starting at time $t$

1. Only N-Servers that are within one N-Server period $T_s$ from their absolute deadline $d_s$ are eligible for scheduling, i.e., the time remaining before the absolute deadline (the laxity) must be less or equal to the
7.3 Simple Server-Scheduled CAN (S\textsuperscript{3}-CAN)

N-Server's relative deadline \(D_s\). Hence, the following condition must hold for an N-Server \(s\) to be eligible for scheduling:

\[
d_s - t \leq D_s
\]  \hspace{1cm} (7.8)

2. Among the eligible N-Servers, select N-Servers in EDF order to be scheduled during the EC.

7.3.3 Timing analysis

As the scheduling mechanism used by Server-CAN is relying on EDF, looking at a specific N-Server, it is normal that an N-Server might have to wait for other N-Servers to be scheduled before it will be scheduled by the M-Server. Since the EDF scheduling performed by the M-Server always treats N-Servers as if they have messages ready for transmission, an arbitrary N-Server \(s\) might be the only N-Server in the system having messages ready for transmission, yet N-Server \(s\) may have to wait for all other N-Servers to be scheduled due to their earlier absolute deadlines. Due to the dynamic behaviour of S\textsuperscript{3}-CAN, proving its worst-case response-time is more complex than the simple PS\textsuperscript{2}-CAN shown in Section 7.2. Actually, for S\textsuperscript{3}-CAN, due to the nature of the N-Server absolute deadline updating strategy in the absence of message transmissions (Equation 7.7), the M-Server scheduler can be in two modes: normal mode and backlog mode. For PS\textsuperscript{2}-CAN, only normal mode exists which makes its worst-case response-time analysis much easier.

**Definition 7.3.1** (Normal mode). The system is in normal mode when all N-Servers' absolute deadlines have values greater than or equal to current time, i.e., \(\forall i : d_i \geq t\).

**Definition 7.3.2** (Backlog mode). The system is in backlog mode when there exists one or more N-Servers with absolute deadlines less than current time, i.e., \(\exists i : d_i < t\).

When the system is in normal mode bandwidth is lost due to Equation 7.7, as an EC terminated earlier than intended is causing the protocol messages (TM and STOP messages) to arrive more frequent, i.e., consume more bandwidth compared to the optimal case when all scheduled N-Servers have messages to transmit.

On the other hand, when the system is in backlog mode bandwidth is gained due to Equation 7.7, as an EC is terminated earlier and the scheduled
N-Server’s absolute deadline is updated to a value greater than what would have been given by Equation 7.6 (since \( t > d_s \)), reclaiming bandwidth that could have been used by the N-Server for message transmissions.

In the following both the normal mode and the backlog mode will be investigated in detail and a corresponding worst-case response-time is derived. In the end, the general (no matter of system mode) worst-case response-time is derived.

### 7.3.4 Worst-case response-time in normal mode

As an example of the \( S^3 \)-CAN scheduler’s run-time performance, consider the set of N-Servers in Table 7.1. After each EC the M-Server is updating the N-Server absolute deadlines accordingly. If all scheduled N-Servers have mes-
7.3 Simple Server-Scheduled CAN ($S^1$-CAN) 149

sages to transmit, the actual run-time schedule is depicted in Figure 7.9a. However, looking at Figure 7.9b, where N-Server 1 has no message to send before time $t$, since its N-Server absolute deadline $d_1$ initially is earlier compared to the other N-Server absolute deadlines, N-Server 1 will be scheduled for message transmission by the M-Server, hence blocking the other N-Servers from message transmission (from being scheduled by the M-Server). Every time N-Server 1 is scheduled but do not have any messages ready for transmission (do not transmit any messages) there will be an increase of protocol overhead, i.e., the number of TM and STOP messages increase. The result of this protocol overhead is (in the example) a late scheduling of N-Server 3 as depicted in Figure 7.9b.

Looking at the example, it can be concluded that an arbitrary N-Server $s$ may in some cases not be scheduled until the EC which overlaps its absolute deadline $d_s$. Since, at this point each of the other N-Servers in the system may have absolute deadlines arbitrary close to (and before) $d_s$, N-Server $s$ may have to wait for each of the other N-Servers to transmit one (1) message. During this period no N-Server will be scheduled twice, since the first time one of the N-Servers are scheduled (at this point with absolute deadlines $d^{n+1}_i$ arbitrary close to $d_s$), its new absolute deadline $d^{n+1}_i$ will be set to a time (value) greater than $d_s$. Hence, the message to be transmitted by N-Server $s$ may in the worst-case be the $|N|$-th message to be sent after time $d_s$.

Determining the worst-case response-time consists of three parts depicted in Figure 7.10: (1) the N-Server $s$ is not ready once scheduled which can cause a delay denoted $T_{\text{not ready}}^s$, (2) the N-Server might be blocked by lower period N-Servers for a duration of $T_{\text{blocked}}^s$, and (3) the N-Server might have to wait $T_{\text{queued}}^s$ time units for all other N-Servers to transmit their messages. However, as will be shown, the system might build up a backlog that also has to be taken into consideration. Below, each of these three parts are investigated and their corresponding temporal characteristics are proven, before finally put together forming the worst-case response-time $R_s$ for a single message transmission through an arbitrary N-Server $s$.

The “not ready” delay

If the N-Server has no messages in its local queue once it has decoded the TM, it will not send any messages during the EC even if messages would arrive at the N-Server during the EC. This is to make sure that scheduled message transmissions will stay within an EC and not overlap to the next EC (as could happen with a late transmission of a message within an EC). All scheduled
N-Servers send their messages to their corresponding CAN communications adapters right after the decoding of the EC. As the M-Server will not see any message transmission of a scheduled N-Server when its message arrived at the N-Server during the EC, the M-Server will assume that no messages exists at the scheduled N-Server and update its N-Server parameters as $d_n^s = \max(t + D_s, d_n^{s-1})$ according to Equation 7.7. Hence, as the N-Server might not be scheduled until the very end of its newly updated N-Server absolute deadline, a delay of $T_{EC} - T_{TM}$ must be added to the N-Server period contributing to the bound on its worst-case response-time.

**Lemma 7.3.3.** An arbitrary N-Server $s$ can have its message transmitted at time $D_s + T_{EC} - T_{TM}$ time units after its arrival at the N-Server. Hence, the worst-case penalty $T_s^{not\_ready}$ for an N-Server not being ready once it is scheduled is given by

$$T_s^{not\_ready} = T_{EC} - T_{TM}$$

**Proof.** Once the message is scheduled by the M-Server and not ready once the TM is decoded at the N-Server, it can in the worst case arrive at the N-Server $T_{EC} - T_{TM}$ time units before the termination of the EC by the transmission and reception of the STOP message at time $t$. Here, at time $t$, the N-Server’s
absolute deadline $d_s$ will be updated to a value of $d'^n_s = \max(t + D_s, d_s^{n-1})$ according to Equation 7.7. Hence, a message will in the worst case be transmitted at a time of $D_s + T_{EC} - T_{TM}$ time units after its arrival at the N-Server, i.e., the penalty for an N-Server not being ready once it is scheduled is given by $T_{\text{not ready}}^s$ in Equation 7.9.

$T_{\text{not ready}}^s$ and its components are depicted in Figure 7.11, where a message arrives right after the decoding of the TM and at time $t$ the N-Server absolute deadline is updated to $d'^n_s = t + D_s$.

![Figure 7.11: Parts composing the worst-case value of $T_{\text{not ready}}^s$.](image)

The “blocking” delay

During the period time $T_s$ of an arbitrary N-Server $s$, any other shorter period N-Server $i$ might be scheduled multiple times. If these shorter period N-Servers have no messages to send, they will have their absolute deadlines continuously updated to a time of $t + D_i$ according to Equation 7.7, where $t$ is the time of scheduling at the M-Server and $D_i$ is the shorter period N-Server’s relative deadline. If the number of these shorter period N-Servers is big enough to fill up one EC (i.e., to fill a whole schedule), these shorter period N-Servers can totally block the system and therefore also block N-Server $s$ from being scheduled. This blocking of N-Server $s$ can continuously occur until a time when the blocking N-Servers’ absolute deadlines will be updated to a value greater than the absolute deadline of N-Server $s$, i.e., an N-Server $i$ can block...
N-Server $s$ as long as $d_s \leq d_i$. This blocking behaviour by shorter period N-Servers is affecting the bound of the worst-case response-time of N-Server $s$.

**Definition 7.3.4 (Blocking).** When a shorter period N-Server interferes with a longer period N-Server by being scheduled for message transmission but not transmitting any message, the shorter period N-Server is blocking the longer period N-Server. This behaviour by the shorter period N-Server may occur multiple times during the period of the longer period N-Server.

**Lemma 7.3.5.** In determining the blocking effect on an arbitrary N-Server $s$, if there exists one shorter period N-Server in the system, it is safe to assume that there exist enough shorter period N-Servers to totally fill up an EC, i.e., to totally fill up a schedule.

**Proof.** As the assumption is pessimistic it is totally safe to assume that there exist enough shorter period N-Servers in the system to totally fill up an EC, even if the number of shorter period N-Servers is less, as this assumption creates a scenario worse than the real scenario.

**Lemma 7.3.6.** In determining the blocking effect on an arbitrary N-Server $s$, it is safe to assume that the N-Server period of all blocking N-Servers have the same length $b$ as the shortest period N-Server (in the system) expressed by

$$b = \min_{i \in N} (T_i) \quad (7.10)$$

**Proof.** As the assumption is pessimistic it is totally safe to assume that all blocking N-Servers have the same N-Server period as the shortest period N-Server among all N-Servers in the system, as captured by $b$ in Equation 7.10. This assumption is safe as longer period blocking N-Servers $i$ cause less or equal blocking (in the worst-case) than shorter period blocking N-Servers $j$, as longer period N-Servers $i$ will at an earlier or equal (not later) time have their absolute deadlines $d_i$ updated by the M-Server to a value greater than $d_s$ (since $d_s - T_i < d_s - T_j$), i.e., compared to shorter period N-Servers, longer period N-Servers will in general sooner (never later) have an absolute deadline with a value greater than the absolute deadline of the N-Server they are blocking.

**Lemma 7.3.7.** An arbitrary N-Server $s$ might, in the worst-case, be blocked by shorter period N-Servers (if shorter period N-Servers exists) for a duration of $T_{s\text{ blocked}}$ captured by

$$T_{s\text{ blocked}} = \sum_{i \in N} (T_i) \quad (7.10)$$
7.3 Simple Server-Scheduled CAN (S\(^3\)-CAN)

\[ T_{\text{blocked}}^s = T_s - b = T_s - \min_{i \in \mathbb{N}} (T_i) \]  \hspace{1cm} (7.11)

**Proof.** It is safe to assume that there exists enough blocking N-Servers to fill one EC (to fill one schedule), and that the N-Server period of all blocking N-Servers are generalised to the shortest period N-Server among all N-Servers in the system, since both these assumptions are pessimistic. This is true since according to Lemma 7.3.5 if there exist one shorter period blocking N-Server there exists enough shorter period N-Servers in the system to totally block N-Servers. Moreover, according to Lemma 7.3.6 the N-Server period of these shorter period blocking N-Servers are all the same as the shortest period N-Server (captured by \( b \) in Equation 7.10). Hence, as all blocking N-Servers will have their absolute deadlines updated to a value greater than \( d_s \) after \( T_s - b \) time units, an arbitrary N-Server \( s \) may be totally blocked for the duration captured by \( T_{\text{blocked}}^s \) in Equation 7.11. \( \square \)

\( T_{\text{blocked}}^s \) and its components are depicted in Figure 7.12, where N-Server \( s \) is blocked until time \( t \) where all blocking N-Servers will get their absolute deadlines updated to a value greater than \( d_s \).
The “queuing” delay

For an arbitrary N-Server \( s \), the worst-case scenario that its blocking N-Servers can perform is to transmit a message as the last event before their corresponding absolute deadlines will be updated to a value greater than \( d_s \). By doing this, the blocking N-Servers will interfere as much as possible with the scheduling of N-Server \( s \). Hence, after the blocking given by \( T_{\text{blocked}} \) in Equation 7.11, the worst-case scenario that can happen to an arbitrary N-Server \( s \) is that all other N-Servers in the system (not only the shorter period blocking N-Servers) have their absolute deadlines right before \( d_s \), and therefore will be scheduled for message transmission before N-Server \( s \), and they will all transmit messages.

Definition 7.3.8 (Queuing delay). At the time when an arbitrary N-Server can not suffer from blocking, it can still have to wait for all N-Servers in the system to transmit one message each. This time is called the queuing delay.

Lemma 7.3.9. In the worst-case scenario, an arbitrary N-Server \( s \) will be the last N-Server to be scheduled among all N-Servers in the system.

Proof. When using the EDF scheduling policy, due to bad phases between the N-Servers in the system, an arbitrary N-Server \( s \) might be scheduled last among all N-Servers in the system since its corresponding absolute deadline \( d_s \) has the highest value (i.e., it has the latest deadline).

Lemma 7.3.10. The worst-case queuing delay \( T_{\text{queued}}^s \) for an arbitrary N-Server \( s \) is given by

\[
T_{\text{queued}}^s = \left\lceil \frac{|N|}{|EC|} \right\rceil \times T_{EC} - T_{STOP}
\]

(7.12)

Proof. The worst-case queuing delay is generated by all N-Servers transmitting messages at the same time, since if some N-Servers would not transmit their messages the ECs where these N-Servers were scheduled for message transmission would be terminated earlier and the total queuing delay would be shorter. If the number \( |N| \) of N-Servers in the system is greater than the size (in messages) \( |EC| \) of an EC, the transmission of all N-Server messages will be spread in a series of ECs where each EC fit \( |EC| \) messages, and N-Server \( s \) will transmit its message in the last EC (according to Lemma 7.3.9). Since \( |N| \)
messages will be transmitted during the queuing delay, and since an arbitrary message in the worst-case will be transmitted as the last message within an EC (according to Lemma 7.1.2), Equation 7.12 captures the worst-case queuing delay, where $T_{STOP}$ removes a small source of pessimism as the last message transmitted in the last EC actually is a STOP message (terminating the EC).

$T_{queued}$ and its components are depicted in Figure 7.13, where N-Server $s$ is blocked until time $t$ where all blocking N-Servers will get their absolute deadlines updated to a value greater than $d_s$. Then N-Server $s$ is subject to a queuing delay, being the last of all N-Servers to be scheduled. Also, once scheduled, the message of N-Server $s$ is the last message to be transmitted in the EC.

**Worst-case response-time**

In normal mode, the worst-case response-time $R_{normal}^s$ for a single message transmitted through an arbitrary N-Server $s$ is retrieved by adding all the above-mentioned delay bounds together.
Theorem 7.3.11. An upper bound on the response-time \( R_s^{\text{normal}} \) for a single message arriving at an arbitrary N-Server \( s \) is

\[
R_s^{\text{normal}} \leq T_s^{\text{not ready}} + T_s^{\text{blocked}} + T_s^{\text{queued}}
\]

\[
= T_{EC} - T_{TM} + T_s - \min_{i \in \mathcal{N}} (T_i) + \left\lceil \frac{|\mathcal{N}|}{|EC|} \right\rceil \times T_{EC} - T_{STOP}
\]

(7.13)

where \( T_s^{\text{not ready}} \) is given by Equation 7.9, \( T_s^{\text{blocked}} \) is given by Equation 7.11, \( T_s^{\text{queued}} \) is given by Equation 7.12.

Proof. The N-Server not ready delay expressed by \( T_s^{\text{not ready}} \) is proven in Lemma 7.3.3, the worst-case blocking time expressed by \( T_s^{\text{blocked}} \) is proven in Lemma 7.3.7 and the queuing delay expressed by \( T_s^{\text{queued}} \) is proven by Lemma 7.3.10. As all delays represent worst-case scenarios, summarising them together produce the worst-case response-time for a single message transmitted through an arbitrary N-Server \( s \). However, as they all are depending on the relative phases between N-Servers, the users sending messages and the EC periodicity, it is likely that not all delays will yield their worst-case values at the same time. Hence, Equation 7.13 provides an upper bound on the response-time for a single message transmitted through N-Server \( s \).

For simplicity Equation 7.13 can be rewritten and both \( T_{TM} \) and \( T_{STOP} \) can safely (pessimistically) be removed (for readability) forming the following inequality for the worst-case response-time for a single message transmitted through an arbitrary N-Server \( s \)

\[
R_s^{\text{normal}} < T_s + \left( 1 + \left\lceil \frac{|\mathcal{N}|}{|EC|} \right\rceil \right) \times T_{EC} - \min_{i \in \mathcal{N}} (T_i)
\]

(7.14)

7.3.5 Worst-case response-time in backlog mode

The worst-case scenario given by Theorem 7.3.11 can cause a relative phase between the N-Servers in the system where all N-Servers, but N-Server \( s \), are scheduled for message transmission before the message transmission of N-Server \( s \). The duration of these message transmissions lasts in the worst-case for \( T_s^{\text{queued}} \) time units (according to Lemma 7.3.10). All N-Servers have an N-Server period shorter than \( R_s^{\text{normal}} \) time units, and are therefore halted during this period in the sense that messages arriving to them as intended with
7.3 Simple Server-Scheduled CAN (S\(^3\)-CAN)

Figure 7.14: Absolute deadlines less than current time.

The period of the N-Server period, will be queued at the N-Server waiting to be transmitted. Note that only one message is transmitted by N-Server \( s \) during the \( R_s^{normal} \) time units. The number of queued messages, \( h_s \), is given by

\[
h_s = \left\lfloor \frac{R_s^{normal}}{T_s} \right\rfloor
\]  

(7.15)

as \( R_s^{normal} \) starts with the arrival of the first message at N-Server \( s \). The question to be answered is if the number of queued messages at an arbitrary N-Server can be bounded by \( h_s \) given by Equation 7.15 or if it will grow beyond \( h_s \).

As the N-Servers are scheduled during the time interval of \( T_s^{queued} \) time units, all N-Servers will initially be served once and their absolute deadlines will be updated accordingly. However, for N-Servers with N-Server periods shorter than \( T_s^{queued} \) time units, at the end of the \( T_s^{queued} \) time units (denoted \( t_q = t \), where \( t \) is the current time at the end of the \( T_s^{queued} \) time units), these absolute deadlines will have values less than the current time \( t_q \). Moreover, these N-Servers with absolute deadlines less than current time might have a number of messages queued in the N-Server.

Consider the scenario depicted in Figure 7.14. Here the duration of \( T_s^{queued} \) time units is ended at time \( t_q \), leaving a number of absolute deadlines with a value less than \( t_q \) (denoted \( d_{a7} \) in the figure), and a number of absolute deadlines with a value greater than \( t_q \) (denoted \( d_{b7} \) in the figure). Now, the question is how these “negative” absolute deadlines will affect the system.
Since the scheduling policy used is EDF, the corresponding N-Servers of the negative absolute deadlines \( d_{a_i} \) will be the first N-Servers to be scheduled.

**Definition 7.3.12.** An absolute deadline with a value less than current time \( t \) is called a negative absolute deadline.

**Definition 7.3.13.** When there exists negative absolute deadlines at current time \( t \), the system is said to have a backlog, i.e., \( \exists i, i \in N : d_i < t \).

**Lemma 7.3.14.** If the system has a backlog, when a scheduled N-Server with a negative absolute deadline does not transmit a message, it is causing the system to gain lost bandwidth.

*Proof.* If the scheduled N-Server with a negative absolute deadline does not transmit a message, its new absolute deadline will be updated to \( d_{a_i}^n = \max(t + D_s, d_{a_i}^{n-1}) \) by Equation 7.7, i.e., a value that for sure is greater than if it would have transmitted a message, giving \( d_{a_i}^n = d_{a_i}^{n-1} + D_s \) by Equation 7.6, since \( d_s < t \) as the system is in backlog mode. By doing this update of N-Server absolute deadline, all bandwidth that the N-Server could have claimed while its absolute deadline was negative is lost for the N-Server and gained for the system. Hence, the system is gaining lost bandwidth. \( \square \)

**Lemma 7.3.15.** If the system has a backlog, the worst-case behaviour by an N-Server with a negative absolute deadline is to always transmit messages when scheduled.

*Proof.* When a scheduled N-Server with a negative absolute deadline does transmit a message its absolute deadline will be updated to \( d_{a_i}^n = d_{a_i}^{n-1} + D_s \) by Equation 7.6, i.e., it might still be negative and it is for sure a value less than \( d_{a_i}^n = \max(t + D_s, d_{a_i}^{n-1}) \) given by Equation 7.7, as \( d_s < t \) during backlog mode. Opposed to what is shown in Lemma 7.3.14, no bandwidth is gained (unless the transmitted message was of less than worst-case size which would cause some slack to be reclaimed). Hence, among the two scenarios of transmitting or not transmitting a message when scheduled, the former is the worst-case behaviour. \( \square \)

**Lemma 7.3.16.** N-Servers with negative absolute deadlines can not cause interference worse than the normal transmission of messages. Hence, the blocking presented in Lemma 7.3.7 (with a result of loosing bandwidth) can not happen when the system has a backlog.
Proof. For an N-Server \( s \) to be blocked there must exist shorter period blocking N-Servers as shown in Lemma 7.3.7. If an N-Server \( s \) has a negative absolute deadline and there exist shorter period N-Servers that potentially could block N-Server \( s \), these N-Servers also have negative absolute deadlines. However, when the N-Servers' absolute deadlines are negative, Lemma 7.3.15 shows that the worst thing these N-Servers can do is to transmit a message. Lemma 7.3.14 shows that when the system has a backlog, not transmitting a message is better (from the system’s point of view as bandwidth is reclaimed) than transmitting a message. Hence, blocking with a result of losing bandwidth can not occur as long as the system has a backlog. As long as there are negative absolute deadlines in the system (i.e., the system has a backlog), bandwidth will either be used by the transmission of messages or it will be reclaimed by the absence of message transmissions or messages being shorter than worst-case size. Hence, bandwidth can not be lost by badly behaving N-Servers when the system has a backlog.

Lemma 7.3.17 (Shifted delay). In a system with a backlog, if all scheduled N-Servers will behave according to their worst-case behaviour (as shown by Lemma 7.3.15) and transmit their messages from now on (from time \( t_q \)), the system will in the worst-case be statically “shifted” with a delay of \( W_s \) time units given by

\[
W_s = T_{\text{queued}}^s - b = \left\lceil \frac{|N|}{|EC|} \right\rceil \times T_{EC} - T_{STOP} - \min_{i \in N}(T_i) \quad (7.16)
\]

where \( T_{\text{queued}}^s \) (given by Equation 7.12) is the worst-case queuing delay built up after a period of blocking and \( b \) is the shortest period N-Server in the system given by Equation 7.10.

Proof. In the worst-case the theoretical load by the N-Servers is configured to a maximum, and their message transmissions will occupy all available bandwidth at all times (as they all will transmit messages which in the worst-case would have worst-case size). As there will always be messages ready for transmission at the N-Servers when scheduled by the M-Server, all messages will be sent according to EDF even though some absolute deadlines might be negative. This since the absolute deadlines will be updated as usual to \( d_n^s = d_{n-1}^s + D_s \) according to Equation 7.6 and what is important from a scheduling point of view is their values relative to the values of the other absolute deadlines, i.e.,
that one absolute deadline is before the other etc. If the absolute deadlines are positive (i.e., value greater or equal to \( t_q \)) or negative (i.e., value less than \( t_q \)) does not matter. Subtracting \( b \) from \( T_{\text{queued}} \) removes the time (during an arbitrary N-Server’s period \( T_s \)) when the queued messages are starting to be transmitted. Hence, Equation 7.16 represents the time left (after an arbitrary N-Server’s period \( T_s \)) needed to transmit the remaining queued messages in the system. The system will in this case be “shifted” with a delay of \( W_s \) time units.

An example of a “shifted” system is given in Figure 7.15, where the normal system is depicted in Figure 7.15a and the system that is shifted \( W_s \) time units is depicted in Figure 7.15b.

**Lemma 7.3.18.** The shifted delay presented by Lemma 7.3.17 will decrease and reach value 0 when there are scheduled N-Servers not transmitting their messages.

**Proof.** As scheduled N-Servers with negative absolute deadlines do not transmit their messages, bandwidth is gained (Lemma 7.3.14) and shared among the
7.3 Simple Server-Scheduled CAN (S$^3$-CAN)

other N-Servers which cause the shifted delay to decrease. This gained bandwidth causes the number of negative absolute deadlines to become smaller and smaller which causes the actual value of $W_s$ (note that the value given by Equation 7.16 is the worst-case initial value of $W_s$) to decrease until all negative absolute deadlines are positive and $W_s = 0$.

Lemma 7.3.19. The shifted delay presented by Lemma 7.3.17 will decrease and reach value 0 when the total theoretical load by the N-Servers in the system is less than the system’s theoretical maximum fulfilling Inequality 7.2.

Proof. If the worst-case theoretical load by the N-Servers in the system is configured less than the system’s maximum, there will be slack in the system where it can recover, i.e., the number of negative absolute deadlines will decrease which cause $W_s$ to decrease until all negative absolute deadlines are positive and $W_s = 0$.

Lemma 7.3.20. The shifted delay presented by Lemma 7.3.17 will decrease and reach value 0 when the actual size of the messages transmitted by the scheduled N-Servers are of less than worst-case size.

Proof. This is true for the same reasons as for Lemma 7.3.18.

Lemma 7.3.21. The number of queued messages $h_s$ will never grow to a value greater than its initial value given by Equation 7.15.

Proof. As pointed out by Lemma 7.3.17 the system will in the worst case be statically shifted with a delay of $W_s$ time units. However, as the system is scheduled according to EDF with a worst-case scenario of scheduled N-Servers to transmit messages (Lemma 7.3.15), messages will be transmitted according to EDF once every N-Server period with a theoretical feasible bandwidth requirement (as the total load of the N-Servers in the system must be a feasible load fulfilling Inequality 7.2). Hence, starting from time $t_q$ all N-Servers will be scheduled for message transmission once every N-Server period (rounded up to the nearest multiple of $T_{EC}$) guaranteeing the number of queued messages at the N-Servers not to grow. As soon as a scheduled message is not
transmitted (or a transmitted message is of less than worst-case size) bandwidth will be reclaimed (Lemma 7.3.14) and the number of queued messages at the N-Servers in the system will decrease over time.

**Theorem 7.3.22.** When the system is in backlog mode, the worst-case response-time $R_{s}^{\text{backlog}}$ of a single message transmitted during this period is captured by

$$R_{s}^{\text{backlog}} \leq T_s + W_s \quad (7.17)$$

where $T_s$ is the N-Server period, $W_s$ is given by Equation 7.16.

**Proof.** As long as the system has a backlog there can not be any bandwidth lost and therefore it is safe to state that the worst-case response-times of the message transmissions during this period is captured by $R_{s}^{\text{backlog}}$. As messages are transmitted the system’s shifted delay $W_s$ (given by Lemma 7.3.17) will either stay the same (when all scheduled messages in an EC are transmitted and have the worst-case message size) or decrease until it reaches the value 0, making the system to leave backlog mode and enter normal mode again as there are no more negative absolute deadlines in the system. This is shown by Lemma 7.3.18, Lemma 7.3.19 and Lemma 7.3.20.

Hence, during backlog mode, based on Inequality 7.17 and with $T_{STOP}$ safely (pessimistically) removed from Equation 7.16, the worst-case response-time $R_{s}^{\text{backlog}}$ for a single message transmitted through an arbitrary N-Server $s$ is given by

$$R_{s}^{\text{backlog}} = T_s + \left\lceil \frac{|N|}{|EC|} \right\rceil \times T_{EC} - \min_{i \in N}(T_i) \quad (7.18)$$

### 7.3.6 Worst-case response-time for $S^3$-CAN

In conclusion, both when the system has no backlog and when it has a backlog, it is safe to say that the worst-case response-time is captured by Equation 7.14 as Theorem 7.3.22 shows that the worst-case response-time with a backlog (given by Equation 7.18) is not worse than the worst-case response-time without a backlog.

**Theorem 7.3.23.** The worst-case response-time for a single message arriving at an arbitrary N-Server $s$, no matter system mode, is given by
7.3 Simple Server-Scheduled CAN (S^3-CAN)

\[ R_s = T_s + \left( 1 + \left\lceil \frac{|N|}{|EC|} \right\rceil \right) \times T_{EC} - \min_{i \in N} (T_i) \quad (7.19) \]

Proof. Since the worst-case response-time given by Equation 7.18 (given by Theorem 7.3.22) is always less than the worst-case response-time given in Equation 7.19 (given by Theorem 7.3.11). This is easily verified as

\[ P^\text{backlog}_s = T_s + \left\lceil \frac{|N|}{|EC|} \right\rceil \times T_{EC} - \min_{i \in N} (T_i) \]
\[ < \]
\[ P^\text{normal}_s = T_s + \left( 1 + \left\lceil \frac{|N|}{|EC|} \right\rceil \right) \times T_{EC} - \min_{i \in N} (T_i) \quad (7.20) \]

Hence, the worst-case scenario for an arbitrary N-Server is given by Theorem 7.3.23, and the worst-case response-time for a single message arriving at an arbitrary N-Server is given by Equation 7.19. The worst-case scenario is now shown in an example.

Note that the implication of Equation 7.19 is a worst-case response-time that is (essentially) dependent on the N-Server period together with the number of N-Servers in the system, hence not only the N-Server deadlines and periodicity. This is unfortunate, since one of the motivations for introducing the N-Servers is to provide isolation between message streams. However, the worst-case bound requires a highly unlikely combination of N-Server phases together with a rather unlikely message arrival pattern at the N-Servers.

Example

As an example following the above reasoning, consider the same set of N-Servers as in Table 7.1. The analytically worst-case scenario for N-Server 3 is depicted in Figure 7.16 where \(|N| = 3\). N-Server 3 is not ready once scheduled and its message arrives right after the decoding of the TM. Then, N-Server 1 is totally blocking the network for \(T^\text{blocked}_3 = T_3 - T_1\) time units, and N-Server 2 has an absolute deadline \(d_2\) slightly before the absolute deadline of N-Server 3 \(d_3\). At time \(t\) all three N-Servers have messages to send and the absolute deadline of N-Server 3 is the latest, hence N-Server 3 has to wait for the other N-Servers to be scheduled before being scheduled itself, as expected by Equation 7.19.
164 Server-CAN schedulers

7.4 Periodic Polling scheduler (PP-CAN)

The Periodic Polling scheduler (PP-CAN) is implemented as a Server-CAN scheduler, intended to be used as a reference scheduler mimicking variants of periodic schedulers, e.g., FTT-CAN [5, 6].

PP-CAN works basically the same way as PS²-CAN, but with the slack reclaiming mechanism disabled, i.e., there is no STOP message terminating the EC. Instead a timer at the M-Server keeps the EC duration to a constant $T_{EC}$, i.e., the duration of the EC is fixed when using PP-CAN.

7.4.1 Updating N-Server states

As with PS²-CAN, when using PP-CAN the M-Server will always treat an N-Server as if it actually did transmit a message (regardless whether it did
transmit any message). Hence the rule for updating the N-Server absolute deadline is always

\[ d^n_s = d^{n-1}_s + D_s \]  \hspace{1cm} (7.21)

Also, as with PS\(^2\)-CAN, the M-Server applies the same rules when deciding the schedule for the EC starting at time \( t \), i.e., only selecting eligible N-Servers in EDF order. The “eligible restriction” is here in order for an N-Server to not be scheduled more than once during a time-span of its N-Server period (see details of how this is done for PS\(^2\)-CAN above).

### 7.4.2 Timing analysis

The worst-case response-time for a single message when using PP-CAN is given by

\[ R_s = 2 \times T_s + T_{EC} \]  \hspace{1cm} (7.22)

following the same reasoning as done for PS\(^2\)-CAN above. However, since PP-CAN has no bandwidth reclaiming mechanism, its average timing performance is expected to be somewhat worse compared to PS\(^2\)-CAN. This is evaluated in the next chapter.

### 7.5 Summary

In this chapter two Server-CAN schedulers are presented, PS\(^2\)-CAN and S\(^3\)-CAN, for which also associated timing analysis is presented and analytically proven.

The first Server-CAN scheduler, PS\(^2\)-CAN, is a dynamic priority bandwidth reclaiming Polling Server (PS) [111, 190, 195] implementation that gives fair bandwidth isolation and responsiveness among the N-Servers in the system. The strength of such a server implementation is its simplicity.

The second Server-CAN scheduler, S\(^3\)-CAN, is a more complex server implementing a bandwidth reclaiming polling version of the Total Bandwidth Server (TBS) [197, 199] trying to be more responsive in its polling of the N-Servers. S\(^3\)-CAN is updating its N-Server parameters similar as the deadline assignment of the TBS. However, depending on how the N-Servers are configured with respect to their actual run-time behaviour (i.e., actual message load during run-time), a badly configured set of N-Servers can for S\(^3\)-CAN
introduce blocking which potentially could cause a waste of bandwidth (and therefore affect the worst-case response-time that can be guaranteed for the N-Servers). However, to produce the worst-case scenarios for $S^3$-CAN require a highly unlikely combination of message arrival patterns and N-Server period phases to occur (as can be read by the assumptions made in the Lemmas and Theorems of deriving the worst-case response-time for $S^3$-CAN).

Also, at the end of this chapter a CAN scheduler is presented that is to be used for evaluation purposes in Chapter 8. This scheduler, PP-CAN, represent other higher layer schedulers available for CAN.
Chapter 8
Evaluation of Server-CAN schedulers

The main objective of this chapter is to evaluate the performance of the two Server-CAN schedulers presented in Chapter 7, comparing them against each other, and against a reference CAN scheduler. Hence, three different CAN schedulers are evaluated with respect to their temporal performance, i.e., the evaluated property is the response-time of message transmissions. The CAN schedulers investigated in this chapter are PP-CAN, PS\(^2\)-CAN and S\(^3\)-CAN, i.e., the reference CAN scheduler named Periodic Polling scheduling mechanism (PP-CAN) presented in Chapter 7 and the two Server-CAN schedulers described in Chapter 7 named Periodic Server-Scheduled CAN (PS\(^2\)-CAN) and Simple Server-Scheduled CAN (S\(^3\)-CAN). PP-CAN is representing other higher layer schedulers available for CAN.

To evaluate PP-CAN, PS\(^2\)-CAN and S\(^3\)-CAN a series of simulations are performed. By using simulation, the same scenarios can be evaluated for all three CAN schedulers, allowing for a fair comparison. The simulation study conducted in this chapter consists of two parts:

- Firstly, the temporal performance of PP-CAN, PS\(^2\)-CAN and S\(^3\)-CAN is investigated in detail, and their specific temporal properties are discussed.
- Secondly, the temporal performance of all schedulers is compared with each other, highlighting strengths and weaknesses of the temporal performances among the schedulers.
The simulator used, for the evaluations in this chapter, is an event-driven simulator that we have developed using C++.

8.1 Simulation set up

For the evaluation of the CAN schedulers, an example system is set up. This system, called the reference system, consists of 40 N-Servers with different N-Server periods. The N-Server periods (in the reference system) are chosen to cover a wide span, including the theoretically shortest N-Server period of $T_s = T_{EC}$, as well as longer N-Server periods. Composing the reference system, a total of 5 N-Server periods are selected with values set to $T_s = a \times \{1, 3, 7, 11, 13\} \times T_{EC}$, where $a$ is a constant used to vary the system load.

To vary the system load, 4 discrete steps are chosen as 99.33%, 75%, 50% and 25%. Also, a special attention is given to the extremes by also looking at 90% and 10%. Hence, a total of 6 discrete steps are investigated for the system load, given by the constant $a$ in the N-Server period $T_s = a \times \{1, 3, 7, 11, 13\} \times T_{EC}$, where $a$ is given by

$$a \in \{1.0, 1.1145, 1.3663, 2.1905, 5.525, 63.5\} \quad (8.1)$$

Also, at the initiation of each simulation, all N-Servers are given a random initial phase $t_{phs}$, randomly selected using a uniform distribution within the value range $t_{phs} \in [0, T_s)$.

Each N-Server $s$ is configured such that it has one associated user. All users transmit messages to the N-Servers in a periodical fashion. The period of the user $T_{user}$ is set to its associated N-Server’s N-Server period $T_s$ as $T_{user} = b \times T_s$, where $b$ is a constant used to vary the server load.

For the server load the following steps are chosen: 100%, 90%, 75%, 50%, 25% and 10%, given by the constant $b$ in the user period $T_{user} = b \times T_s$, where $b$ is given by

$$b \in \{1.0, 1.1111, 1.3333, 2.0, 4.0, 10.0\} \quad (8.2)$$

Also, at the initiation of each simulation, all users are given a random initial phase $t_{phu}$, randomly selected using a uniform distribution within the value range $t_{phu} \in [0, T_{user})$. Moreover, each message generated by any user has a unique randomly generated message identifier. Hence, for each user period the message identifier of the user message will be different.

In order to investigate the temporal performance in various system configurations, the system load (the load by the N-Servers on the system) and the
server load (the load by the users on the N-Servers) is varied in a number of discrete steps between high and low. The system load is varied between 99.33% and 10% by the constant $a$ in the N-Server periods $T_s$, and the server load is varied between 100% and 10% by the constant $b$ in the user periods $T_{user}$. Hence, a total of 6 discrete steps are investigated in two dimensions by varying both the system load and the server load.

Each simulation configuration (with a specific system load and server load, as well as randomised N-Server and user phases) is simulated 1000 times for which the worst-case measured response-times are registered for each N-Server. Also, the average-case response time of all 1000 simulations is calculated.

Each simulation is run for 100000 ms. The N-Server periods are between 4.43 ms (for the shortest N-Server period in the simulations with 99.33% system load) and 3656.965 ms (for the longest N-Server period in the simulations with 10% system load). The user periods are between 4.43 ms (for the shortest user period in the simulations with 99.33% system load and 100% server load) and 36569.65 ms (for the longest user period in the simulations with 10% system load and 10% server load).

Looking at the N-Server periods, they can have 5 different values. Throughout this chapter the N-Servers with the shortest period ($a \times 1 \times T_{EC}$) are denoted N-Server type "1", and the N-Servers with the longest period ($a \times 13 \times T_{EC}$) are denoted N-Server type "5". In the reference system, there are 10 N-Servers of type "1", 8 N-Servers of type "2" ($a \times 3 \times T_{EC}$), 7 N-Servers of type "3" ($a \times 7 \times T_{EC}$), 5 N-Servers of type "4" ($a \times 11 \times T_{EC}$) and 10 N-Servers of type "5". Hence, in the simulations there are a total of 40 N-Servers. The maximum system load caused by these 40 N-Servers are 99.33%, with $a = 1.0$.

For the case with 99.33% system load, in Figure 8.1, the 5 types (shown as 1 to 5) of N-Servers are shown as their N-Server periods together with their corresponding theoretical worst-case response-times, for PP-CAN and PS$^2$-CAN, and S$^3$-CAN respectively. Note that as the N-Server periods become longer (due to decreasing of system load), the worst-case response-times for PP-CAN and PS$^2$-CAN will have a similar relation to their corresponding N-Server period as in Figure 8.1, whereas the worst-case response-times for S$^3$-CAN will approach the value of their corresponding N-Server period (as the worst-case response-time for S$^3$-CAN, apart from the N-Server period, is constant). This is shown in Figure 8.2, where the worst-case response-times for PP-CAN and PS$^2$-CAN approaches 2.0 as the system load goes down, whereas the worst-case response-times for S$^3$-CAN approaches 1.0. Figure 8.2 shows, for PP-CAN and PS$^2$-CAN, and S$^3$-CAN, the worst-case response-times for
Evaluation of Server-CAN schedulers

Figure 8.1: Periods ($\alpha = 1.0$) and analytical worst-case response-times for PP-CAN, PS$^2$-CAN and S$^3$-CAN for all 5 N-Server types.

Figure 8.2: Normalised analytical worst-case response-times for all 5 N-Server types in the 6 simulated system load scenarios (99.33% - 10%).
8.2 Simulation result format

As the reference system is simulated, what are shown in the figures throughout this chapter are not the exact values of all 40 N-Servers but the processed values

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network speed (bits/ms)</td>
<td>500</td>
</tr>
<tr>
<td>M size (bits)</td>
<td>135</td>
</tr>
<tr>
<td>TM size (bits)</td>
<td>135</td>
</tr>
<tr>
<td>STOP size (bits)</td>
<td>55</td>
</tr>
<tr>
<td>$T_M$ (ms)</td>
<td>0.27</td>
</tr>
<tr>
<td>$T_{TM}$ (ms)</td>
<td>0.27</td>
</tr>
<tr>
<td>$T_{STOP}$ (ms)</td>
<td>0.11</td>
</tr>
<tr>
<td>$</td>
<td>EC</td>
</tr>
<tr>
<td>$T_{EC}$ (ms)</td>
<td>4.43</td>
</tr>
<tr>
<td>$T_{sched}$ (ms)</td>
<td>0</td>
</tr>
<tr>
<td>$</td>
<td>N</td>
</tr>
<tr>
<td>$T_s$ (ms)</td>
<td>$a \times {1, 3, 7, 11, 13} \times T_{EC}$</td>
</tr>
<tr>
<td>$T_s$ (ms)</td>
<td>$1.0, 1.1145, 1.3663, 2.1905, 5.525, 63.5$</td>
</tr>
<tr>
<td>$T_{s_{\text{max}}}$ (ms)</td>
<td>$4.43 \times T_s = a \times 1 \times T_{EC}, a = 1.0$</td>
</tr>
<tr>
<td>$T_{s_{\text{min}}}$ (ms)</td>
<td>$3656.965 \times T_s = a \times 13 \times T_{EC}, a = 63.5$</td>
</tr>
<tr>
<td>$l_{phs}$ (ms)</td>
<td>$t_{phs} \in [0, T_s)$ (uniform distribution)</td>
</tr>
<tr>
<td>Number of users</td>
<td>40 (one to each N-Server)</td>
</tr>
<tr>
<td>$T_{user}$ (ms)</td>
<td>$T_s \times b$</td>
</tr>
<tr>
<td>$b$</td>
<td>${1.0, 1.1111, 1.3333, 2.0, 4.0, 10.0}$</td>
</tr>
<tr>
<td>$T_{user_{\text{max}}}$ (ms)</td>
<td>$4.43 \times T_{user} = T_{user_{\text{min}}} \times 1.0$</td>
</tr>
<tr>
<td>$T_{user_{\text{min}}}$ (ms)</td>
<td>$36569.65 \times T_{user} = T_{user_{\text{max}}} \times 10.0$</td>
</tr>
<tr>
<td>$l_{phu}$ (ms)</td>
<td>$t_{phu} \in [0, T_{user})$ (uniform distribution)</td>
</tr>
<tr>
<td>Simulation time (ms)</td>
<td>$10,000$</td>
</tr>
<tr>
<td>Number of simulations</td>
<td>1000</td>
</tr>
</tbody>
</table>

Table 8.1: Simulation parameters.

the 5 different N-Server types (shown as 1 to 5) as the system load goes from 99.33% (the leftmost graph), 90%, 75%, 50%, 25% to 10% (the rightmost graph). In the figure, the worst-case response-times are normalised with their corresponding N-Server’s period for the sake of readability.

All the parameters of the simulation study are shown in Table 8.1.
of the 5 types of N-Servers (shown as 1, 2, 3, 4 and 5 in the figures). The values for each N-Server type are obtained by taking the mean value of all N-Servers with N-Server periods matching the corresponding N-Server type. Also, the actual measured response-times are not shown, but the normalised measured response-times. The response-times are normalised against their corresponding N-Server’s N-Server period. This is done for the sake of readability, to manage all information obtained in the simulation study.

### 8.3 Evaluation of PP-CAN

In this section the Periodic Polling scheduling mechanism (PP-CAN) is evaluated. First, the temporal expectations are discussed, followed by the simulation results.

#### 8.3.1 Expectations

Using PP-CAN, each N-Server will be periodically scheduled according to EDF. According to EDF, an N-Server can be scheduled at any moment within its N-Server period as the time of scheduling depends on the competing deadlines in the system, which in turn can be arbitrary phased. Hence, the longest distance between two scheduled instances of N-Server \( s \) is \( \sim 2 \times T_s \), as the
8.3 Evaluation of PP-CAN

N-Server $s$ might be scheduled for message transmission at the very beginning of its N-Server period followed by being scheduled for message transmission at the very end of its next N-Server period. This scenario is depicted in Figure 8.3.

Since deadlines might occur inside an EC, it is natural that some messages will not be transmitted until after a time of $2\times T_s + T_{EC}$ time units, i.e., there can be a violation of the N-Server’s absolute deadline $d_s$ by $T_{EC}$ time units. The reason for this is that time is partitioned into ECs and the Server-CAN framework does not control exactly when a specific message is sent inside an EC due to the CAN arbitration mechanism (as discussed in Section 6.3.2 and formally proven by Lemma 7.1.2).

Looking at the worst-case response-time for PP-CAN, depicted in Figure 8.4, N-Server $s$ is scheduled for message transmission (Figure 8.4a) and a message is arriving (Figure 8.4c) at the N-Server $s$ close to the beginning of its N-Server period $T_s$. However, no message is ready for message transmis-
174 Evaluation of Server-CAN schedulers

sion in N-Server $s$ once the TM is decoded (Figure 8.4b), so the message will not be transmitted during the EC. Once the EC is terminated the N-Server’s absolute deadline is updated to $d_{n+1} = d_n + D_s$ (Figure 8.4d), and the N-Server is not scheduled again until the EC containing $d_{n+1}$ (Figure 8.4e). Once the N-Server is scheduled again, its message is the last message to be transmitted in the EC (Figure 8.4f). Hence, the expected (in the simulation study) worst-case response-time for PP-CAN is $2 \times T_s + T_{EC}$, which is also formally proven in Chapter 7.

8.3.2 Simulation results

In this section, the results of the simulations are presented. First, the worst-case measured response-times are discussed followed by the discussion of the average-case measured response-times.

Worst-case measured response-times

The results of the worst-case measured response-times for PP-CAN are shown for a system load of 99.33%, 90% and 75% in Figure 8.5 and for a system load of 50%, 25% and 10% in Figure 8.6. In both figures, the server load goes from 100% to 10% (indicated as 1.0 to 0.1 in the figures) for each system load.

Looking at the worst-case measured response-times for PP-CAN, apart for N-Server type “1”, the measured response-times are quite close, i.e., have similar values. This is expected in the PP-CAN simulations, as the N-Server period should not affect the worst-case measured response-times too much (apart from N-Servers with short N-Server periods such as N-Server type “1”).

During high system load the N-Servers are expected to have a worst-case value close to 2.0 (derived from $2 \times T_s$). In general, when the N-Server periods are larger than $T_{EC}$, the penalty of $T_{EC}$ is a less important part of the total response-time and does not clearly show in the figures as the values in the figures are normalised against the N-Server period.

The N-Server type “1” has an N-Server period of $a \times T_{EC}$, which is the shortest N-Server period among all N-Servers in the simulation study. Specifically, when the system load is high, it is likely that the distance between two scheduled instances of the N-Server is $2 \times T_s$ due to N-Servers competing to be scheduled. Also, the penalty of $T_{EC}$, as the message can be transmitted as last message in the EC, is quite visible. Therefore, when the system load is 99.33% ($T_s = 1.0 \times T_{EC}$) the worst-case measured response-times are close to 3.0 ($2 \times T_s + T_{EC} = 3 \times T_{EC}$). As the system load goes down, the worst-
Figure 8.5: Worst-case measured response-times for PP-CAN with 99.33%, 90% and 75% system load as the server load goes from 100% to 10%.

Figure 8.6: Worst-case measured response-times for PP-CAN with 50%, 25% and 10% system load as the server load goes from 100% to 10%.
Case measured response-times for N-Servers type “1” first approaches 2.0, as the impact of the $T_{EC}$ penalty becomes less visible when the N-Server periods are longer (when the system load becomes lower). In the end, as only eligible N-Servers are scheduled, only few N-Servers will be scheduled in each schedule (as the system load is low). Due to this, all N-Servers are scheduled within their respective N-Server period, resulting in worst-case measured response-times that are approaching 1.0. Hence, PP-CAN is behaving as expected in the worst-case.

Average-case measured response-times

The results of the average-case measured response-times for PP-CAN are shown for a system load of 99.33%, 90% and 75% in Figure 8.7 and for a system load of 50%, 25% and 10% in Figure 8.8. In both figures, the server load goes from 100% to 10% (indicated as 1.0 to 0.1 in the figures) for each system load.

Looking at the average-case measured response-times, again N-Server type “1” behaves differently compared to the other types. The same reason as discussed for the worst-case measured response-times are valid here, i.e., this is due to the short N-Server period of N-Server type “1”.

Another observation is that when the server load is 100%, the average-case measured response-times are higher compared with all other simulations with lower server load (but with the same system load). The reason for this is that when an N-Server is fully loaded at 100% it is likely that the phases between the user and the N-Server will not be arbitrary but static for the duration of one simulation, causing a static shift between the message arrivals at the N-Server and the scheduling of the N-Server (as both are periodic events with fixed periods). This static shift is only present when the server load is exactly 100% (given by $b = 1.0$), i.e., the user period has the same value as the N-Server period $T_{user} = T_s$ and the N-Server has a static phase $t_{phs}$ given a random value within $[0, T_s)$ at the start of the simulation, and the user has a static phase $t_{phu}$ given a random value within $[0, T_{user})$, also at the start of the simulation. Any other user period $T_{user} \neq T_s$ makes this static shift less likely.

When the server load goes down from 100%, the average-case measured response-times stabilises around 0.5, which is expected as the user messages are arriving at the N-Servers in a periodic manner, and the N-Servers and users can have an arbitrary phase between each other. Hence, it is likely that the average-case measured response-times will be half of the N-Server period, i.e., 0.5 in the figures. PP-CAN is behaving as expected in the average-case.
8.3 Evaluation of PP-CAN

Figure 8.7: Average-case measured response-times for PP-CAN with 99.33%, 90% and 75% system load as the server load goes from 100% to 10%.

Figure 8.8: Average-case measured response-times for PP-CAN with 50%, 25% and 10% system load as the server load goes from 100% to 10%.
8.4 Evaluation of PS\(^2\)-CAN

In this section the Periodic Server-Scheduled CAN scheduling mechanism (PS\(^2\)-CAN) is evaluated. First, the temporal expectations are discussed, followed by the simulation results.

8.4.1 Expectations

The difference between PP-CAN and PS\(^2\)-CAN is that PP-CAN has fixed size ECs whereas PS\(^2\)-CAN does not have fixed size ECs. The N-Server parameters are updated the same way for both schedulers. However, whenever there are no more messages transmitted in an EC, PP-CAN wait until the fixed length end of the EC when PS\(^2\)-CAN terminates the EC to make use of (reclaim) the unused bandwidth. Hence, using PS\(^2\)-CAN, compared to PP-CAN, bandwidth is reclaimed and therefore the temporal performance of PS\(^2\)-CAN is expected to be slightly better than PP-CAN, since there will be more bandwidth available in the system, especially in the simulations with high system load.

8.4.2 Simulation results

In this section, the results of the simulations are presented. First, the worst-case measured response-times are discussed, followed by a discussion of the average-case measured response-times.

Worst-case measured response-times

The results of the worst-case measured response-times for PS\(^2\)-CAN are shown for a system load of 99.33%, 90% and 75% in Figure 8.9 and for a system load of 50%, 25% and 10% in Figure 8.10. In both figures, the server load goes from 100% to 10% (indicated as 1.0 to 0.1 in the figures) for each system load. As expected, and shown in the figures, the worst-case measured response-times behave similar to PP-CAN.

Average-case measured response-times

The results of the average-case measured response-times for PS\(^2\)-CAN are shown for a system load of 99.33%, 90% and 75% in Figure 8.11 and for a system load of 50%, 25% and 10% in Figure 8.12. In both figures, the server load goes from 100% to 10% (indicated as 1.0 to 0.1 in the figures) for each system load. As expected, and shown in the figures, the average-case measured response-times behave similar to PP-CAN.
8.4 Evaluation of PS\(^2\)-CAN

Figure 8.9: Worst-case measured response-times for PS\(^2\)-CAN with 99.33%, 90% and 75% system load as the server load goes from 100% to 10%.

Figure 8.10: Worst-case measured response-times for PS\(^2\)-CAN with 50%, 25% and 10% system load as the server load goes from 100% to 10%.
Figure 8.11: Average-case measured response-times for $PS^2$-CAN with 99.33%, 90% and 75% system load as the server load goes from 100% to 10%.

Figure 8.12: Average-case measured response-times for $PS^2$-CAN with 50%, 25% and 10% system load as the server load goes from 100% to 10%.
8.5 Evaluation of S$^3$-CAN

In this section the Simple Server-Scheduled CAN scheduling mechanism (S$^3$-CAN) is evaluated. First, the temporal expectations are discussed, followed by the simulation results.

8.5.1 Expectations

As the N-Servers are not synchronised with the system users, a bad phase between users, N-Servers and ECs influence the temporal performance of the message transmissions when using S$^3$-CAN. Several factors contribute to degrading the temporal guarantees possible with S$^3$-CAN, which are presented and discussed in this section.

The N-Server is not ready when scheduled

Firstly, as the ECs are not synchronised with the N-Servers it is natural that a message will arrive at an N-Server at any time within an EC. It is not certain that a message is ready for transmission before the start of an EC, as assumed by the M-Server once it schedules an N-Server. In the worst-case, a message that is scheduled for transmission within an EC will arrive at the N-Server right after the N-Server has received and decoded a TM containing the schedule saying that it has been scheduled for message transmission this EC. Since the N-Server has no message at the time, it will not send any messages this EC and must wait until the next time it is scheduled for message transmission, regardless if the message arrives right after the decoding of the TM.

This scenario is depicted in Figure 8.13 where the scheduled N-Server $s$ (Figure 8.13a) have no messages ready for transmission once scheduled (Figure 8.13b). Therefore, even though a message arrives at the N-Server during the EC (Figure 8.13c), the N-Server is not allowed to transmit a message during the intended EC causing the M-Server to update its absolute deadline $d^n_s$ to $d^n_{s+1} = t + D_s$ (Figure 8.13d). Hence, in the worst-case the N-Server might not be scheduled until a time of $D_s = T_s$ time units after current time $t$ (due to its newly updated absolute deadline). The penalty for not being ready once scheduled is denoted $T_{not\,\,ready}^s$, given by $T_{not\,\,ready}^s = T_{EC} - T_{TM}$ time units.

A late transmission within an EC

As when using PP-CAN and PS$^2$-CAN, since absolute deadlines might be located inside an EC, it is natural that some messages will not be delivered until
Evaluation of Server-CAN schedulers

Figure 8.13: No message ready once the TM is decoded will delay its transmission to a later EC.

\[ T_s + T_{EC} - T_{STOP} \] time units, i.e., there can be a violation of the deadline \( d_s \) by \( T_{EC} - T_{STOP} \) time units. The reason for this is that \( S^3 \)-CAN (and the other Server-CAN schedulers) does not control exactly when a specific message is sent inside an EC due to the CAN arbitration mechanism (as discussed in Section 6.3.2).

**Approach to expected worst-case response-time**

Putting these two scenarios together indicate that a message can be transmitted as late as \( T_s + 2 \times T_{EC} - T_{TM} - T_{STOP} \) time units after its arrival at the N-Server. This is due to a scenario caused by a bad phase between the initiation of the EC and the message arriving at the N-Server, and a bad phase between the initiation of a later EC containing the N-Server’s absolute deadline \( d_s \) and the actual location of \( d_s \) within this EC.

Such a scenario is depicted in Figure 8.14. Here, bad phases give a worst-case scenario for a message arriving at N-Server \( s \). The message arrives slightly after the time when N-Server \( s \) decodes the TM (Figure 8.14c). Since the
8.5 Evaluation of $S^3$-CAN

A scheduled N-Server has no message ready for transmission at the time when N-Server $s$ decodes the TM (Figure 8.14b), it will not transmit any messages within that EC. This is monitored by the M-Server, updating the absolute deadline for N-Server $s$ from $d_{n}^{s}$ to $d_{n+1}^{s} = t + D_s$ (Figure 8.14d), where $t$ is the time of scheduling (i.e., the time of the termination of the EC where the message arrived at N-Server $s$). Now, a high load by the system’s N-Servers cause N-Server $s$ to be scheduled as late as possible (Figure 8.14e). This, together with the N-Server’s absolute deadline to be located within an EC, and that the message is transmitted as late as possible (Figure 8.14f) within this EC (due to the CAN arbitration mechanism) cause a worst-case scenario. Hence, looking at a single N-Server, a message transmitted through an arbitrary N-Server $s$ can under some bad circumstances be transmitted as late as $T_s + 2 \times T_{EC} - T_{TM} - T_{STOP}$ time units after its arrival at the N-Server.
What about interference?

However, so far an N-Server \( s \) has only been investigated in isolation. In fact, the other N-Servers in the system also contribute to the worst-case scenario for N-Server \( s \). Since the system is scheduled based on absolute deadlines according to EDF, all N-Servers with absolute deadlines that are earlier than \( d_s \) are scheduled before N-Server \( s \). This should not be a problem as long as the system is feasible according to the EDF rules, i.e., the total system load should be less than 100%. However, as messages might not be ready at an N-Server once scheduled by the M-Server, scheduling N-Servers with no messages ready for transmission consumes bandwidth and will influence the worst-case response-time of N-Server \( s \). In fact, all N-Servers with a shorter period than that of N-Server \( s \) can be scheduled multiple times during a time interval of \( T_s \), i.e., during the N-Server period time of N-Server \( s \). If these N-Servers have no messages to transmit, they can cause all bandwidth in the system to be consumed by the Server-CAN protocol messages as depicted in Figure 8.15, and they can do this until the time when their absolute deadlines will be updated to a value higher than \( d_s \). This time is defined as \( T_s^{\text{blocked}} \). A safe upper bound on \( T_s^{\text{blocked}} \) is \( T_s^{\text{blocked}} = T_s \), i.e., the network will not be transmitting any messages but protocol messages for the duration of the N-Server period of N-Server \( s \). However, this assumption (bound) is very pessimistic and a less pessimistic bound can easily be derived as

\[
T_s^{\text{blocked}} = T_s - \min_{i \in \mathcal{N}} (T_i)
\]

(8.3)

where \( \mathcal{N} \) is the set of all N-Servers in the system. This bound is valid since a blocking attempt on N-Server \( s \) (by a shorter period blocking N-Server), at any time later than \( T_s^{\text{blocked}} \) time units after the initiation of the N-Server’s period \( T_s \), will cause the blocking N-Servers’ absolute deadlines to be updated to a value greater than \( d_s \). Hence, after this time the blocking N-Servers will not contribute more to the scenario depicted in Figure 8.15. Note that this less pessimistic bound still assumes that the whole EC is filled with (scheduled for) blocking N-Servers and that the blocking N-Servers must have a relative phase (in terms of N-Server periods) of the worst kind. Hence, the bound is safe but very unlikely to occur in a real scenario.

The worst thing an N-Server can do to another N-Server is to block it as described above, causing the Server-CAN protocol messages to consume all available bandwidth. However, this type of blocking can only be done by N-Servers that have N-Server periods shorter than the N-Server they are blocking. This since N-Servers with periods longer than the N-Server that they are
8.5 Evaluation of $S^3$-CAN

Bandwidth consumed by the protocol due to scheduling of N-Servers with no messages ready for transmission

Here the blocking N-Servers’ absolute deadlines will be updated to a value higher (greater) than that of the absolute deadline of N-Server $s$

Figure 8.15: N-Servers with shorter N-Server periods can consume all bandwidth.

blocking will have their absolute deadlines updated to a value greater than the N-Server they are blocking as soon as the M-Server is updating their N-Server parameters. So, the worst thing these N-Servers can do (the N-Servers with N-Server periods longer than the N-Server they are blocking) to the N-Server they are blocking is to transmit their messages, consuming as much bandwidth as they possible can. Therefore, apart from blocking an N-Server as described above, the worst thing an arbitrary N-Server can do to another N-Server is to transmit its message, as the transmission of N-Server messages will consume as much bandwidth as possible before the blocked N-Server will be allowed to transmit its message.

Hence, the worst-case scenario that can happen to N-Server $s$, after all bandwidth has been consumed by protocol messages as depicted in Figure 8.15, is that all other N-Servers in the system (including the ones that just blocked N-Server $s$) have their absolute deadlines just before time $d_s$ and are therefore scheduled before the scheduling of N-Server $s$. Also, all these N-Servers have messages ready to be transmitted. If there are $|N|$ N-Servers in the system, N-Server $s$ will in the worst-case be transmitted as the last message among all messages that are to be transmitted, i.e., after a time $T_{s^{\text{opened}}}$ captured by
Evaluation of Server-CAN schedulers

\[ T_s^{\text{queued}} = \left\lceil \frac{|\mathcal{N}|}{|EC|} \right\rceil \times T_E - T_{STOP} \quad (8.4) \]

where \(|\mathcal{N}|\) is the number of N-Servers in the system, \(|EC|\) is the size of the EC in number of messages that it fits (worst-case size messages), and \(T_{STOP}\) is the time taken to transmit the STOP message (which will be the last message transmitted in the last EC).

This transmission of messages will start after a time of \(T_s^{\text{not\-ready}} + T_s^{\text{blocked}}\) time units after the arrival of the message at the N-Server \(s\). Note that Equation 8.4 also covers for the scenario with a late transmission within an EC presented above.

Again, this bound assumes a worst-case phase between the N-Server periods of all interfering N-Servers causing their absolute deadlines to be located just before \(d_s^n\). Hence, the bound is safe but very unlikely to occur in a real scenario (i.e., in a properly designed system).

Worst-case response-time

Combining the above reasoning, the worst-case scenario is captured in Figure 8.16, where the message originally scheduled (Figure 8.16a) arrives at the N-Server (Figure 8.16c) after the decoding of the TM (Figure 8.16b), which cause the N-Server’s absolute deadline to be updated from \(d_s^n\) to \(d_s^{n+1}\) (Figure 8.16d) and not scheduled again until a later EC (Figure 8.16e), finally transmitting the message as the last message within the EC (Figure 8.16f).

Hence, using S\(^3\)-CAN, the following analytical expression capture the worst-case response-time \(R_s\) for an arbitrary message transmitted through an arbitrary N-Server \(s\)

\[ R_s = T_s^{\text{not\-ready}} + T_s^{\text{blocked}} + T_s^{\text{queued}} \]

\[ = T_E - T_{TM} + T_s - \min_{i \in \mathcal{N}} (T_i) + \left\lceil \frac{|\mathcal{N}|}{|EC|} \right\rceil \times T_E - T_{STOP} \quad (8.5) \]

Expected worst-case response-time in these simulations

However, as the simulation study is set up, messages are periodically produced by the users, and hence, it is likely that the pessimistic pattern by the blocking N-Servers (causing blocking followed by a transmission of a message, all in worst-case phase with the other blocking N-Servers) will not happen. It is more likely that Equation 8.4 will behave as
8.5 Evaluation of $S^3$-CAN

\[ T_{\text{queued}} = T_{EC} - T_{STOP} \quad (8.6) \]

as it is likely that after the blocking scenario depicted in Figure 8.15, only the blocked N-Server $s$ has a message ready for transmission.

Hence, the expected worst-case response-times in these simulations are more likely to be as depicted in Figure 8.14 as it is common for an N-Server to have no messages ready for transmission in the simulation study, in the simulations with low server load. The expected worst-case response-times are captured by

\[ R_s = T_s - \min_{i \in N} (T_i) + 2 \times T_{EC} - T_{TM} - T_{STOP} \quad (8.7) \]
8.5.2 Simulation results

In this section, the results of the simulations are presented. First, the worst-case measured response-times are discussed followed by the discussion of the average-case measured response-times.

Worst-case measured response-times

The results of the worst-case measured response-times for S³-CAN are shown for a system load of 99.33%, 90% and 75% in Figure 8.17 and for a system load of 50%, 25% and 10% in Figure 8.18. In both figures, the server load goes from 100% to 10% (indicated as 1.0 to 0.1 in the figures) for each system load.

Looking at the temporal performance of S³-CAN it has an interesting behaviour compared to PP-CAN and PS²-CAN. As the server load goes down, the worst-case measured response-times goes down also for S³-CAN (as for PP-CAN and PS²-CAN), but only for N-Server type ”1” and ”2”. For N-Server type ”3”, ”4” and ”5” the worst-case measured response-times goes up as the server load decreases. This is due to the blocking caused by the N-Server type ”1” (mostly) and N-Server type ”2” (some). When the server load goes down, it is more likely that there are no messages ready for transmission when the N-Servers are scheduled. Hence, the blocking behaviour described above and depicted in Figure 8.15 is likely to occur. This behaviour is affecting the temporal performance of the longer period N-Servers type ”3”, ”4” and ”5” causing the result as shown in the figures. Hence, it is important that the N-Servers in the system are configured properly to avoid this type of blocking (interference). The blocking causes blocked N-Servers to be scheduled late in their N-Server period, which is verified in the figures as the worst-case measured response-times for blocked N-Servers approach 1.0. The expected value is $T_{\text{blocked}}$ (which is a value less than 1.0), given the N-Server period of the blocking N-Servers (N-Server type ”1” and ”2”) which in the simulations should be $3 \times T_{EC}$ when the system is fully loaded, as the longest N-Server period among the 15 (size of the EC schedule) shortest N-Server period N-Servers has an N-Server period of $a \times 3 \times T_{EC}$. N-Server type ”1” and ”2” are not blocked as they are the N-Servers causing the blocking. In conclusion, S³-CAN is behaving as expected looking at the worst-case measured response-times.
8.5 Evaluation of S\textsuperscript{3}-CAN

Figure 8.17: Worst-case measured response-times for S\textsuperscript{3}-CAN with 99.33%, 90% and 75% system load as the server load goes from 100% to 10%.

Figure 8.18: Worst-case measured response-times for S\textsuperscript{3}-CAN with 50%, 25% and 10% system load as the server load goes from 100% to 10%.
Average-case measured response-times

The results of the average-case measured response-times for $S^3$-CAN are shown for a system load of 99.33%, 90% and 75% in Figure 8.19 and for a system load of 50%, 25% and 10% in Figure 8.20. In both figures, the server load goes from 100% to 10% (indicated as 1.0 to 0.1 in the figures) for each system load.

For the average-case measured response-times similar behaviour as in the worst-case measured response-times are found, which is as expected, and the same reasoning as for the worst-case measured response-times can be made. However, note the low values of the average-case measured response-times compared to the corresponding values when using PP-CAN or $PS^2$-CAN.

8.6 Comparing PP-CAN, $PS^2$-CAN and $S^3$-CAN

A comparison of the temporal performance of PP-CAN, $PS^2$-CAN and $S^3$-CAN is shown for worst-case measured response-times for a system load of 99.33% in Figure 8.21, 90% in Figure 8.23, 75% in Figure 8.25, 50% in Figure 8.27, 25% in Figure 8.29 and 10% in Figure 8.31. The average-case measured response-times are shown (with the same corresponding system load as the worst-case measured response-times) in Figure 8.22, Figure 8.24, Figure 8.26, Figure 8.28, Figure 8.30 and Figure 8.32. In all figures, the server load goes from 100% to 10% (indicated as 1.0 to 0.1 in the figures) for each system load.

It can be seen in the figures that PP-CAN and $PS^2$-CAN have similar performance, with a slight advantage for $PS^2$-CAN. This is expected as they both update their N-Server parameters the same way, but the bandwidth reclaiming feature of $PS^2$-CAN should make it slightly more efficient compared with PP-CAN.

What is more interesting is the temporal performance of $S^3$-CAN compared with both PP-CAN and $PS^2$-CAN. $S^3$-CAN provides much better temporal performance no matter the system load, both for the worst-case measured response-times and the average-case measured response-times. This is especially visible when the system load goes down, where the temporal performance of $S^3$-CAN becomes very good, with worst-case measured response-times less than $\sim 0.25$ (down to 0.00125) instead of $\sim 1.0$ as PP-CAN and $PS^2$-CAN and with average-case measured response-times less than $\sim 0.1$ (down to 0.00022) instead of $\sim 0.5$ as for PP-CAN and $PS^2$-CAN, due to the extra bandwidth available in the system that can be used by the Server-CAN protocol to perform as responsive scheduling as possible.
8.6 Comparing PP-CAN, PS²-CAN and S³-CAN

Figure 8.19: Average-case measured response-times for S³-CAN with 99.33%, 90% and 75% system load as the server load goes from 100% to 10%.

Figure 8.20: Average-case measured response-times for S³-CAN with 50%, 25% and 10% system load as the server load goes from 100% to 10%.
Figure 8.21: Worst-case measured response-times for PP-CAN, PS\textsuperscript{2}-CAN and S\textsuperscript{3}-CAN with 99.33% system load as the server load goes from 100% to 10%.

Figure 8.22: Average-case measured response-times for PP-CAN, PS\textsuperscript{2}-CAN, S\textsuperscript{3}-CAN with 99.33% system load as the server load goes from 100% to 10%.
8.6 Comparing PP-CAN, PS\textsuperscript{2}-CAN and S\textsuperscript{3}-CAN

Figure 8.23: Worst-case measured response-times for PP-CAN, PS\textsuperscript{2}-CAN and S\textsuperscript{3}-CAN with 90% system load as the server load goes from 100% to 10%.

Figure 8.24: Average-case measured response-times for PP-CAN, PS\textsuperscript{2}-CAN and S\textsuperscript{3}-CAN with 90% system load as the server load goes from 100% to 10%.
Figure 8.25: Worst-case measured response-times for PP-CAN, PS$^2$-CAN and S$^3$-CAN with 75% system load as the server load goes from 100% to 10%.

Figure 8.26: Average-case measured response-times for PP-CAN, PS$^2$-CAN and S$^3$-CAN with 75% system load as the server load goes from 100% to 10%.
8.6 Comparing PP-CAN, PS\textsuperscript{2}-CAN and S\textsuperscript{3}-CAN

Figure 8.27: Worst-case measured response-times for PP-CAN, PS\textsuperscript{2}-CAN and S\textsuperscript{3}-CAN with 50% system load as the server load goes from 100% to 10%.

Figure 8.28: Average-case measured response-times for PP-CAN, PS\textsuperscript{2}-CAN and S\textsuperscript{3}-CAN with 50% system load as the server load goes from 100% to 10%.
Figure 8.29: Worst-case measured response-times for PP-CAN, PS$^2$-CAN and S$^3$-CAN with 25% system load as the server load goes from 100% to 10%.

Figure 8.30: Average-case measured response-times for PP-CAN, PS$^2$-CAN and S$^3$-CAN with 25% system load as the server load goes from 100% to 10%.
8.6 Comparing PP-CAN, PS\textsuperscript{2}-CAN and S\textsuperscript{3}-CAN

Figure 8.31: Worst-case measured response-times for PP-CAN, PS\textsuperscript{2}-CAN and S\textsuperscript{3}-CAN with 10\% system load as the server load goes from 100\% to 10\%.

Figure 8.32: Average-case measured response-times for PP-CAN, PS\textsuperscript{2}-CAN and S\textsuperscript{3}-CAN with 10\% system load as the server load goes from 100\% to 10\%.
8.7 Summary

This chapter evaluates the temporal performance of PP-CAN, PS²-CAN and S³-CAN, investigating their response-times both analytically (discussion on expected values) and through a set of simulations of a reference system. During the simulations the response-times of the N-Servers in the reference system are measured and compared with their corresponding analytical worst-case response-time values, in order to verify Server-CAN’s temporal performance.

The results of the evaluation show, as expected, that PS²-CAN performs slightly better than PP-CAN. However, more interesting is the temporal performance of S³-CAN, which performs much better than both PP-CAN and PS²-CAN in all simulated scenarios. In the simulations, both system load and server load are varied in order to capture the temporal behaviour of all schedulers under various configurations. In all simulated scenarios, S³-CAN performs better than both PP-CAN and PS²-CAN.

Also, the bandwidth isolation property is tested in the sense that the CAN message identifiers used in the simulations are not configured according to Rate Monotonic, where identifiers (= priorities) are assigned based on the period of the message transmitter. The CAN identifiers used in the simulations are arbitrary, which on a native CAN network would cause deadline violations as CAN is scheduled based on the messages’ priorities. As all the measured response-times are within the analytical worst-case response-times, and as the message identifiers are arbitrary, the bandwidth isolation property is kept when using Server-CAN.
Chapter 9

Using Server-CAN

Traditionally in many application domains, subsystems have been developed incrementally as new functionality has been added to the system. Looking at the automotive domain, value- and safety-adding functions, such as ABS and VDC, have been added over the years. Initially, they could be integrated as (mostly) independent subsystems having their own dedicated hardware in terms of ECUs and communications network. However, as the number of subsystems increases, there are strong trends towards integration of the subsystems on a common distributed architecture instead of using their own. Hence, a crucial issue to solve is the migration from federated systems to integrated systems [103].

Looking at the integration of distributed embedded systems, the network interconnecting the nodes (interconnecting the ECUs in the automotive domain) is a commonly shared resource. As the systems are integrated on the network, they should not interfere in such way that they cause violations of each other’s timing requirements. This can be achieved by the use of a flexible and predictable scheduler to schedule the message transmissions on the network.

In the previous chapters, CAN is identified as the major network technology in the automotive domain and it holds a strong position in other domains as well. Following this, the Server-CAN scheduler (and its variants) has been presented as a flexible scheduling framework implementing server-based scheduling algorithms. In this chapter it is shown how to use the Server-CAN scheduling framework in the context of subsystem integration. Using the Server-CAN

\[^{1}\text{Automotive Open System Architecture (AUTOSAR). http://www.autosar.org/}\]
framework, the major issue to solve in terms of integration is how to translate an existing distributed subsystem to a system using N-Servers.

9.1 Configuring the Server-CAN framework

The issue of configuring the Server-CAN framework consists of two parts. Firstly, it must be decided which Server-CAN scheduler to use (as they have different temporal characteristics). Secondly, all message transmissions in the original system must now pass through a set of N-Servers that must be set up properly. Hence, as an N-Server is allowed to send one message each N-Server period, from a system point of view, a sufficient number of N-Servers must be set up where each N-Server must be configured with an appropriate N-Server period.

9.1.1 Selecting a Server-CAN scheduler

As shown in Chapter 7 and Chapter 8, different Server-CAN schedulers provide different guarantees in terms of timeliness. The two Server-CAN schedulers considered in this chapter are PS\(^2\)-CAN and S\(^3\)-CAN.

Using the PS\(^2\)-CAN scheduler, the analytical worst-case response-time, for a single message transmitted by an arbitrary N-Server \(s\), is given by

\[
R_s = 2 \times T_s + T_{EC} \tag{9.1}
\]

where \(T_s\) is the N-Server period and \(T_{EC}\) is the length of the EC.

Using S\(^3\)-CAN schedulers, the analytical worst-case response-time, for a single message transmitted by an arbitrary N-Server \(s\), is given by

\[
R_s = T_s + \left(1 + \left\lceil \frac{|N|}{|EC|} \right\rceil \right) \times T_{EC} - \min_{i \in N}(T_i) \tag{9.2}
\]

where \(R_s\) is (apart from the N-Server period \(T_s\)) mainly the result of the system configuration in terms of the number of N-Servers in the system \(|N|\), the length of the EC in messages \(|EC|\), the length of the EC in time \(T_{EC}\) and, related to possible blocking, the shortest N-Server period among the other N-Servers in the system \(\min_{i \in N}(T_i)\).

Although S\(^3\)-CAN has shown to provide better temporal performance compared to PP-CAN and PS\(^2\)-CAN (in the simulation study in Chapter 8), it is more sensitive to the system configuration. During integration, the adding of
new subsystems will cause the number of N-Servers in the system to increase and possibly also affect the worst-case possible blocking. As the following two parts of the worst-case response-time for S$^3$-CAN depend on the number and configuration of the other N-Servers in the system

$$\left\lfloor \frac{|\mathcal{N}|}{|EC|} \right\rfloor \quad \text{and} \quad \min_{i \in \mathcal{N}} (T_i)$$

adding a new subsystem might affect the temporal performance of the already integrated parts of the system. Therefore, it may be necessary to reconfigure the parameters of the N-Servers belonging to the already integrated subsystems when a new subsystem is integrated into the system. This does not affect the subsystems directly, but the system integrator (the person responsible for the integration process) must have this in mind when using S$^3$-CAN. Note that this is not a problem when using PS$^2$-CAN. However, the temporal performance of PS$^2$-CAN is not as good as the performance of S$^3$-CAN. Hence, selecting a Server-CAN scheduler is a trade-off depending on the characteristics of the application.

9.1.2 Configuring N-Servers

Definition 9.1.1. A message transmitter, a part of an application or a whole application, is called a user.

Definition 9.1.2. The original system is the system before the usage of the Server-CAN framework.

There are two ways of assigning N-Servers to the users in the original system. Either each user is assigned an N-Server of its own, or several users share one N-Server.

However, as the Server-CAN scheduling framework divides time into ECs, the resolution of time is limited to $T_{EC}$ (i.e., the length of an EC). Due to this, in order for a system to be feasible when using the Server-CAN framework, it must be assumed that each user in the original system can handle this degradation of temporal resolution given by the most responsive N-Server, which will have an N-Server period of $T_{EC}$.

Assumption 9.1.3. It is assumed that each message transmitter (user) in the original system (intending to use the Server-CAN framework) can tolerate the temporal resolution of Server-CAN.
Hence, in order for the system to be feasible, the N-Server periods must be configured such that the theoretical worst-case response-time of the N-Server does not violate the temporal requirements of the user(s) in the original system.

One user in each N-Server

The most obvious way to translate an existing distributed system to a system running the Server-CAN framework would be to create one N-Server for each user. In this way it must (only) be made sure that the temporal guarantees given by the N-Server will fulfill the temporal guarantees required by the original user.

Multiple users in each N-Server

It is not very resource efficient to provide each user with a dedicated N-Server, as the number of N-Servers in the system might have to be very high in this case. The more N-Servers there are in the system, the bigger TM must be used. Using $S^3$-CAN, the analytical worst-case response-time will be dependent on the number of N-Servers in the system, i.e., less N-Servers gives better response-times for all N-Servers in the system (this is not the case for PP-CAN and PS²-CAN). Also, as the theoretical worst-case response-time always is greater than the N-Server period (i.e., $R_s > T_s$ no matter which Server-CAN scheduler is selected), having one periodic user in each N-Server would potentially waste bandwidth since the N-Server period must be made shorter than the original periodic user’s period in order to give the same temporal performance.

A better usage of the network’s bandwidth resource is to let several users of the same subsystem share an N-Server. The sharing is possible when using, e.g., a priority based queue at the N-Server. Then, the sharing users will (within the N-Server) have the same priorities as in the original federated system, and get temporal performance accordingly as long as the bandwidth $Q_s$ and the best worst-case response-time $R_s$, provided by the N-Server, is sufficient. However, when several users are sharing an N-Server, it must be verified that all users will meet their temporal requirements. This can be verified using an appropriate response-time analysis, as presented below.
9.1 Configuring the Server-CAN framework

Busy period

In Chapter 7 the worst-case response-time for a single message sent through an arbitrary N-Server has been investigated and analytically proven for both PS$^2$-CAN and S$^3$-CAN. However, in the context of Server-CAN configuration it is interesting to know how multiple messages sent through an arbitrary N-Server are scheduled and transmitted, as an N-Server can be shared by multiple users.

PS$^2$-CAN can from time to time have messages transmitted with an interval $R_s$ given by Equation 9.1. However, following such an interval is a new interval not exceeding $T_s$, as all N-Servers are scheduled once every N-Server period. In fact, a time less than $T_s$ is more likely.

S$^3$-CAN can be in either normal mode or in backlog mode. In normal mode, all N-Servers are scheduled within their N-Server periods. When entering backlog mode, an N-Server can be scheduled and transmitting its message once within $R_s$ as given by Equation 9.2. However, in backlog mode, S$^3$-CAN will schedule N-Servers more frequently (at least once every N-Server period) before it goes back to normal mode (see details in Chapter 7).

Hence, looking at both Server-CAN schedulers, when scheduling N-Servers during an arbitrary time interval of $t_\Delta > R_s$, an arbitrary N-Server will transmit messages at least $n_s$ times, given by

$$n_s = \left(1 + \left\lfloor \frac{t_\Delta - R_s}{T_s} \right\rfloor \right)$$  \hspace{1cm} (9.3)

In the long run, the N-Server is sometimes scheduled more frequently and sometimes less frequently than $T_s$. However, the two cancels each other as following a less frequent scheduling, when the distance between two scheduled instances is greater than $T_s$, is a scheduling with a distance between two scheduled instances of less or equal to $T_s$. This can, during a time interval $t_\Delta$, be represented as an arbitrary N-Server transmit messages $n_s - 1$ times once every N-Server period $T_s$ and one time within the corresponding $R_s$. Translating this into several users sharing an N-Server, when looking at the response-time for an arbitrary user, this means that all higher priority users will get their messages transmitted once every N-Server period $T_s$ and the user under analysis will get its message transmitted within $R_s$. 
Response-time analysis

Based on the reasoning above, the analytical worst-case response-time for a single message $i$ transmitted through an arbitrary N-Server $s$, when the N-Server is shared with a set of periodic message transmitters, is given by the recurrence relation

$$R_{n+1} = \sum_{j \in hp(i)} \left( \left\lfloor \frac{R_n - T_{EC}}{T_j} \right\rfloor + 1 \right) \times T_s + C_i$$

(9.4)

where

- $C_i$ is the transmission time of message $i$. The transmission time of a single message $i$ is given by $C_i = R_s$, where $R_s$ is the worst-case response-time for N-Server $s$, given by Equation 9.1 for PS²-CAN and Equation 9.2 for S³-CAN.
- $T_{EC}$ caters for the non pre-emptive transmission of the message $i$, as it is not part of the N-Server’s level-$i$ busy period.
- $T_s$ is the N-Server period for N-Server $s$.
- $T_j$ is the period for user $j$.
- $hp(i)$ is the set of tasks with priority higher than that of message $i$.

Allocating N-Servers to nodes

In the optimal case (in terms of using as few N-Servers as possible) it is possible to provide each node in the original (federated) distributed system with a single N-Server of sufficient capacity. Then, each user on that node transmits its messages through that N-Server. As the total bandwidth requirement $Q_n$ for an arbitrary node $n$ with $m$ users is given by

$$Q_n = \sum_{i=0}^{m-1} \frac{T_M}{\min(D_i, T_i)}$$

(9.5)

where $T_M$ is the length (in time) of a message, $D_i$ is the relative deadline of the user, and $T_i$ is the period of the user $i$, an N-Server serving this node must be configured such that it can handle this load.
9.2 Integration of subsystems

As an N-Server can send one message of length $T_M$ each N-Server period, and the bandwidth requirement $Q_n$ for an arbitrary node $n$ is given by Equation 9.5, a sufficient N-Server period $T_s$ is obtained by

$$T_s = \frac{T_M}{Q_n} \quad (9.6)$$

However, as the shortest N-Server period $T_s^\text{min}$ for an arbitrary N-Server $s$ is limited to the length of an EC (i.e., $T_s^\text{min} = T_{EC}$), the maximum bandwidth provided by a single arbitrary N-Server $s$ has an upper bound $Q_s^\text{max}$ given by

$$Q_s^\text{max} = \frac{T_M}{T_{EC}} \quad (9.7)$$

where $T_{EC}$ is the length of the elementary cycle, it is possible that multiple N-Servers must be set up to handle the bandwidth requirement of $Q_n$. Hence, a total of $k_n$ N-Servers must be set up for each node $n$, where $k_n$ is given by

$$k_n = \left\lceil \frac{Q_n}{Q_s^\text{max}} \right\rceil \quad (9.8)$$

and the N-Server periods of these N-Servers must be configured in such a way that the temporal guarantees required by the users that the N-Server is serving are still fulfilled, i.e., by calculating the analytical worst-case response-time for each N-Server and match this against the users served by the N-Server. The analytical worst-case response-time $R_s$ for an arbitrary N-Server $s$ is the best worst-case temporal guarantees that can be offered by the server. Hence, an N-Server $s$ cannot serve a message transmitter with higher temporal requirements than $R_s$. The optimal partitioning of the message transmitters among the N-Servers, and the configuration of the N-Server periods, is a classical bin packing problem [37, 57].

9.2 Integration of subsystems

Above it has been discussed which Server-CAN scheduler to use and it has been shown how to configure the N-Server parameters. Once the N-Server parameters are configured properly, the subsystems will run together without violating any temporal requirements. As the N-Servers provide bandwidth isolation and a predictable worst-case response-time on message transmissions, integrating several subsystems on the same network will not cause any of the N-Servers to violate their temporal guarantees.
In order for subsystem integration using Server-CAN to work (as well as using any other integration approach relying on a CAN network), it is required that within the system there must be no two users using any same message identifier. This restriction applies across subsystems as well, i.e., it is required that all subsystems use unique message identifiers, also among the subsystems.

**Assumption 9.2.1.** It is assumed that within the system, all subsystems use unique message identifiers, i.e., two arbitrary subsystems can not use the same message identifier for any message.

Originally, without using Server-CAN, it is possible that two independently developed subsystems have unique message identifiers. The only requirement put on a subsystem is that it will fulfil its temporal requirements in isolation. However, as the usage of message identifiers is not specified among several subsystems, it is possible that all messages of one subsystem have only higher priority than all messages of the other subsystem. Integrating these two subsystems on the same network will likely cause the subsystem using only low priority messages to violate its temporal requirements.

However, in the real world this is not how subsystems (that are intended to be integrated) are designed. Usually, the system integrator (on system level) provides message identifiers to the subsystem developers based on their initial requirements. Then, these message identifiers are to be used within the subsystem exactly as agreed upon between the system integrator and the subsystem developer. As modern systems are very complex, so is the task of initial allocation of message identifiers (based on initial subsystem designs) and therefore it is likely that early mistakes on message allocation will be discovered only years later. Fixing such problems could be extremely costly as they involve not only the subsystem that needs the change but also other subsystems, as the change affects the other subsystems within the system. In fact, the interdependency between the automotive subsystems is one of the major problems and cost factor in today’s automotive systems [131]. This since any change in one subsystem might change the behaviour of the entire system, requiring difficult and expensive retesting of major parts of the system, as system reliability is of greatest importance. Also, it might not be possible (for the system integrator) to get the complete picture until all subcontractors have delivered their (final) subsystems [172]. Moreover, the subcontractors may only be able to do limited testing themselves, due to the lack of the complete information about the network load characteristics.
9.3 Example

By using the Server-CAN framework, this situation is prevented as the CAN message identifiers are decoupled from the message’s priority. Hence, as the value of the message identifiers will not affect the temporal performance of their corresponding message, the integration process will not suffer from a change in the temporal performance of the subsystems to be integrated, inherited by the actual values of the message identifiers.

9.3 Example

In order to demonstrate the usage of the Server-CAN framework in a "real" case, a small S3-CAN example system is set up, inspired from Chapter 3. In a BMW 7 series messages have periods of 10 ms and higher. Suppose our example subsystem consists of four nodes, where (for simplicity) each node has the same set of users as outlined in Table 9.1. On each node, a single N-Server will be set up for the users, as users sharing an N-Server provide better bandwidth utilisation. Hence, our example subsystem will consist of four N-Servers.

<table>
<thead>
<tr>
<th>id</th>
<th>T</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>4</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>5</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

Table 9.1: User properties.

Looking at the system where the subsystem is to be integrated, let’s assume that there exists a total of 20 N-Servers in the system (including the four to be integrated). The 16 already existing N-Servers in the system require some given bandwidth (not required to be specified for this example), but they will not interfere with the four to be integrated due to the bandwidth isolation property of the Server-CAN framework. Moreover, let’s assume that the size of an EC is configured so that it fits 10 messages, and that the shortest period N-Server in the system (information needed for the worst-case response-time calculations of S3-CAN) has an N-Server period equal to the length of an EC. The system assumptions are outlined in Table 9.2.
Based on the system parameters (network speed and message sizes), using Equation 9.5, it is possible to calculate the total bandwidth requirement $Q_n$ by the set of users on node $n$ outlined in Table 9.1.

$$Q_n = \sum_{i=1}^{5} \frac{T_M}{\min(D_i, T_i)} = 0.03195 \quad (9.9)$$

Running an N-Server provides worst-case response times longer than the N-Server period, as indicated by Equation 9.2. This temporal degradation has to be taken into account when dimensioning the N-Servers. Actually, using S$^3$-CAN the worst-case response time can be represented as

$$R_s = T_s + T_{pen}^{S^3-CAN} \quad (9.10)$$

where $T_{pen}^{S^3-CAN}$ is a static penalty that can be calculated using the system parameters to

$$T_{pen}^{S^3-CAN} = \left(1 + \left\lceil \frac{|N|}{|EC|} \right\rceil \right) \times T_{EC} \times \min_{i \in N} (T_i)$$

$$= (1 + 2) \times 1.54 - 1.54 \quad (9.11)$$

$$= 3.08$$

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network speed (bits/ms)</td>
<td>1000</td>
</tr>
<tr>
<td>M size (bits)</td>
<td>135</td>
</tr>
<tr>
<td>TM size (bits)</td>
<td>135</td>
</tr>
<tr>
<td>STOP size (bits)</td>
<td>55</td>
</tr>
<tr>
<td>$T_M$ (ms)</td>
<td>0.135</td>
</tr>
<tr>
<td>$T_{TM}$ (ms)</td>
<td>0.135</td>
</tr>
<tr>
<td>$T_{STOP}$ (ms)</td>
<td>0.055</td>
</tr>
<tr>
<td>$</td>
<td>EC</td>
</tr>
<tr>
<td>$T_{EC}$ (ms)</td>
<td>1.54</td>
</tr>
<tr>
<td>$T_{sched}$ (ms)</td>
<td>0</td>
</tr>
<tr>
<td>$</td>
<td>N</td>
</tr>
</tbody>
</table>

Table 9.2: System parameters.
Due to this penalty, sending a single message that has a relative deadline of, e.g., 10, through an N-Server (within the message’s relative deadline), requires that the N-Server has a worst-case response time of $R_s = 10$. In other words, since $R_s = T_s + T_{pen}^{S^3-CAN}$, the period of that N-Server must be set to $10 - T_{pen}^{S^3-CAN} = 10 - 3.08 = 6.92$ in order to fulfill the deadline requirement of the message. Looking again at the bandwidth requirement of our example, this corresponds to shortening each relative deadline to $D_i - T_{pen}^{S^3-CAN}$. Hence, it is possible to calculate a sufficient bandwidth $Q_s$ for the N-Server as

$$Q_s = \sum_{i=1}^{5} \frac{T_M}{\min(D_i, T_i) - T_{pen}^{S^3-CAN}} = 0.041263072$$

(9.12)

Since the N-Server is only serving a bandwidth of $Q_n$ as given by Equation 9.9, the N-Server is utilised to $\approx 77\%$. Hence, it is possible to use the rest of the bandwidth capacity of the N-Server for other messages, e.g., non real-time messages (but also real-time messages since the message transmissions are predictable).

To run our example system, an N-Server $s$ has to be configured to meet the bandwidth requirement of $Q_s$. A sufficient N-Server period $T_s$ is given by

$$T_s = \frac{T_M}{Q_s} = 3.271690467$$

(9.13)

Based on this N-Server period, the worst-case response times for the messages sent by the users outlined in Table 9.1 is calculated using Equation 9.2. The worst-case response times are shown in Table 9.3.

<table>
<thead>
<tr>
<th>id</th>
<th>T</th>
<th>D</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>10</td>
<td>6.3517</td>
</tr>
<tr>
<td>2</td>
<td>15</td>
<td>15</td>
<td>9.6234</td>
</tr>
<tr>
<td>3</td>
<td>25</td>
<td>25</td>
<td>19.4385</td>
</tr>
<tr>
<td>4</td>
<td>50</td>
<td>50</td>
<td>25.9818</td>
</tr>
<tr>
<td>5</td>
<td>100</td>
<td>100</td>
<td>39.0686</td>
</tr>
</tbody>
</table>

Table 9.3: Message response times using $S^3$-CAN.

As can be seen in Table 9.3, all message response times have values less than their respective relative deadline, i.e., all deadlines are met. However, note that the allocation of all users to one N-Server, and the setting of the N-Server
period, is just done as an example. Using a more detailed analysis of the system and the users, an optimal user partitioning to N-Servers can be made, as well as an optimal configuration of the N-Server periods. To do this, an appropriate assisting tool should be developed.

9.3.1 Cost

The example above shows a feasible N-Server configuration based on an example system. However, using the Server-CAN framework introduces a cost in reduced temporal performance and overhead.

Best-case worst-case

The most obvious degradation of temporal performance is the division of time into ECs. In the example above, the best-case possible worst-case response time for a single N-Server would be

\[
R_s = T_s + T_{pen}^{S^3-CAN}
\]

\[
= T_{EC} + T_{pen}^{S^3-CAN}
\]

\[
= 1.54 + 3.08
\]

\[
= 4.62
\]

(9.14)

which should be compared to the time required to transmit a single message of highest priority on the CAN network. For the highest priority message this would be

\[
R = B + T_M
\]

\[
= 0.135 + 0.135
\]

\[
= 0.27
\]

(9.15)

assuming that the highest priority message is of worst-case size and that it can be blocked by a lower priority message also of worst-case size.

However, it should be pointed out that the best-case worst-case response time for a single N-Server indicates a time where a total of \(|EC|\) messages are sent, i.e., in our example, at least 10 messages are sent as “fast” as within
4.62\,ms. The same scenario of sending 10 messages on a CAN network without using N-Servers, given that the message identifiers are assigned properly (using the Server-CAN framework they can be arbitrary), would require only 1.485\,ms.

**N-Server utilisation**

As discussed above, having multiple users sharing an N-Server is better compared to configure the system with one N-Server to each user. Suppose that each user is allocated one N-Server. In this case, using the same system parameters as in our example, in order to fulfil the temporal requirements of the users, the resulting N-Servers are not fully utilised. In Figure 9.1, by varying the user period, it is shown:

1. how much bandwidth that is needed by the user, given by

\[
Q_{\text{user}} = \frac{T_M}{T_{\text{user}}}
\]  

\noindent (9.16)
2. how much bandwidth that would be required by a sufficient (fulfilling the user’s relative deadline) N-Server, given by

\[ Q_{N\text{-Server}} = \frac{T_M}{T_{user} - T_{pen,CAN}} \]  (9.17)

Based on Figure 9.1, Figure 9.2 shows how much an N-Server is utilised when it is configured with only one user. The longer the user period is, the more the N-Server is utilised. Note that the unused bandwidth is not lost but can be used for predictable message transmission of other users. What must be made sure is that the total utilisation required by the users of one N-Server is not greater than the utilisation provided by the N-Server. The utilisation required by the users is captured by Equation 9.5.

For example, in our example above, five users are sharing an N-Server and the N-Server is configured to fulfil all users’ temporal requirements. This resulted in an N-Server with a bandwidth of 0.041263072 when only 0.03195 is needed by the users, i.e., the utilisation of the N-Server is \( \approx 77\% \).
9.3 Example

Figure 9.3: Overhead by the Server-CAN protocol.

**Network utilisation**

Looking at the whole system, the maximum utilisation of the network is constrained by the length (in time) of a TM message $T_{TM}$, the length (in time) of a STOP message $T_{STOP}$, the computational overhead (in time) $T_{sched}$ of using the Server-CAN framework, and the length (in time) of an EC $T_{EC}$. As discussed in chapters 6 to 7, the theoretical upper bound on the network utilisation is given by

$$Q^{max} = 1 - \frac{T_{TM} + T_{STOP} + T_{sched}}{T_{EC}}$$  \hspace{1cm} (9.18)

Again, looking at our example system, the theoretical upper bound in our example is

$$Q^{max} = 1 - \frac{0.135 + 0.055 + 0}{1.54}$$

$$\approx 87\%$$  \hspace{1cm} (9.19)
Hence, it is possible to configure our example system with N-Servers of a total network utilisation requirement of \( \approx 87\% \).

Depending on the length of the EC in time, the lower bound on the protocol overhead introduced by the Server-CAN protocol is for our example system depicted in Figure 9.3. As can be seen, the lower bound on the protocol overhead decrease the longer the EC is, which means that the upper bound on the theoretical network utilisation increase the longer the EC is.

### 9.4 Summary

In this chapter it is shown how to use the Server-CAN framework in the context of subsystem integration. It is discussed which Server-CAN scheduler to use, i.e., the difference of the temporal guarantees provided by the Server-CAN schedulers presented in this thesis are illustrated. Moreover, it is shown how to configure the N-Server parameters for a subsystem, and in the end of the chapter, the integration process of subsystems using the Server-CAN concept is discussed, and a small example of how to configure the Server-CAN framework for a “real” subsystem is show. Also, the cost of using the Server-CAN framework, in terms of temporal degradation and overhead, is discussed.
Chapter 10

Summary

This thesis presents a new framework for share-driven schedulers of embedded networks. The framework is shown to facilitate predictable and efficient integration of subsystems relying on the Controller Area Network (CAN).

Looking at CAN, one of the obstacles for a smooth integration of subsystems is the relationship between a message’s identifier and the message’s priority. As the message identifier represents the message priority, and as the CAN network is scheduled based on the messages’ (fixed) priorities, integrating two independently developed subsystems could easily cause interference between the subsystems leading to a situation in which one or several of the integrated subsystems violate their timing requirements [131]. Hence, the management of message identifiers when using a CAN network is a delicate task, especially if the CAN based system comprises many subsystems. When the system consists of multiple independently developed subsystems it might not be possible to know the temporal performance of the complete system until all subsystems are developed and delivered for integration in their final state [172].

The CAN network is very predictable thanks to its MAC protocol, which relies on bitwise arbitration of messages to resolve collisions (CAN is implementing CSMA/CR). The problem (from a subsystem integration point of view) is the connection between a message’s priority and its identity. Moreover, in order to provide hard real-time guarantees, the schedulability bound when using CAN is about 69% when using a Rate Monotonic (RM) priority assignment. This bound is due to the Fixed-Priority Scheduling (FPS) performed by the CAN MAC protocol. This is not good as the bandwidth offered by CAN itself is low, at most 1 Mbps.
Hence, the research potential is twofold. Firstly, it would be beneficial to disassociate the message’s priority from its identity. Secondly, there is a potential to raise the schedulability bound of CAN to 100% by scheduling the messages based on dynamic priorities according to Earliest Deadline First (EDF). In terms of the focus of the thesis, disassociating the message’s priority from its identifier removes a great obstacle when subsystems relying on a CAN network are integrated.

10.1 Research questions revisited

In Chapter 1, an initial question was raised setting the scene of the potential application of the research presented in this thesis:

**Q1 What characterise networked Electronic Automotive Systems (EASs)?**

The nature of this question is very general and to answer it requires it to be broken down into several sub questions. Hence, Q1 is broken down into four sub questions as follows:

1. Which typical EASs exist?
2. What is the main issue with EAS based system development?
3. Which network technologies do these EASs use?
4. How are these network technologies scheduled?

Answering Q1-1 and Q1-2, the thesis begins by surveying EASs in Chapter 3, presenting the evolution of EASs, typical EASs, their requirements, the issue of subsystem integration of EASs, as well as some real EASs and how they are composed (Volvo XC90, BMW 7 Series and VW Passat). This survey is included in the thesis as an example application domain showing some real systems and real requirements to motivate the theoretical work presented later in the thesis (the research contributions presented in chapters 6 to 9).

Following Chapter 3, and answering Q1-3 and Q1-4, network technologies for embedded systems are surveyed in Chapter 4, with a focus on the automotive domain. Also, a brief overview is given of the network technologies used in the avionic domain, the train domain, the industrial and process automation domain and the building and home automation domain. The survey on network technologies is a part of the thesis since the key common shared resource when
integrating distributed embedded subsystems is the network, as the network will be shared among all subsystems.

Supported by the two surveys presented in Chapter 3 and Chapter 4, CAN is identified as one of the major network technologies used in several of these application domains, especially in the automotive domain. Hence, CAN gets a special treatment with a chapter of its own. In Chapter 5, CAN is investigated in detail, surveying standards, real-time analysis, message schedulers and higher layer protocols.

After the abovementioned initial part of the thesis (chapters 2 to 5, where Chapter 2 presents basic concepts and terms used throughout the thesis), the research contributions are presented in chapters 6 to 9. Based on proposition P1 given in Chapter 1, and the answer to question Q1, three specific research questions are formulated:

Q2 How can share-driven scheduling be implemented in the network technologies used by EASs?

Q3 What is the real-time performance of such an implementation?

Q4 Does such an implementation facilitate integration of EASs?

Answering Q2, as an attempt to simplify the subsystem integration process from a network point of view, the Server-CAN scheduling framework is proposed and presented in Chapter 6. The framework allows for share-driven schedulers to be implemented for the CAN network, which is a good scheduling policy from an integration point of view, since the scheduling is performed based on shares (fractions) of a resource, i.e., on an abstraction higher than on a single message basis. The CAN network is selected and used by the framework since it is the most common network technology used by EASs, and it is common and used in many other application domains as well (as presented in chapters 4 to 5).

Following the description of the Server-CAN framework, and answering Q3, in Chapter 7 two Server-CAN schedulers are presented, and their corresponding worst-case response-time for a single message transmitted through an arbitrary N-Server is analytically determined and proven. Also, a reference CAN scheduler is presented, representing other higher layer schedulers for the CAN network. In Chapter 8 all presented CAN schedulers are evaluated in a simulation study, which shows the potential for using Server-CAN schedulers.

Finally, answering Q4, in Chapter 9 it is shown that Server-CAN facilitates subsystem integration from a network point of view (under two realistic as-
218 Summary

sumptions), and it is shown how to use the Server-CAN scheduling framework in the context of subsystem integration.

10.2 Answering the proposition

The proposition stated in Chapter 1 reads as follows:

P1 Share-driven scheduling of event-triggered networks facilitates efficient integration of electronic subsystems from a network point of view.

In the proposition, the property to achieve efficiency is that if subsystems are integrated with other subsystems, they will not interfere with each other. As a consequence, subsystems do not need to know the details of each other. This makes the integration process more efficient, as subsystems can be developed independently, and verifications done on the subsystem in isolation will by design hold also for the integrated system.

As all research questions Q1-Q4 are answered, and as the results of Chapter 9 show, the usage of the Server-CAN scheduling framework facilitates integration of subsystems from a network point of view under two realistic assumptions:

1. **Assumption 9.1.3** - It is assumed that each message transmitter in the original system (intending to use the Server-CAN scheduling framework) can tolerate the temporal resolution of Server-CAN.

2. **Assumption 9.2.1** - It is assumed that within the system, all subsystems use unique message identifiers, i.e., an arbitrary two subsystems can not use the same message identifier for any message.

10.3 Results: The Server-CAN framework

As when using native CAN, the Server-CAN framework provides timely and predictable delivery of messages transmitted on the network. However, running the Server-CAN framework on top of native CAN introduces some new properties. Several strong points motivate the usage of Server-CAN although it all comes at a cost, as Server-CAN has some weak points as well.

The Server-CAN framework provides the following advantages over native CAN in terms of added flexibility:
10.3 Results: The Server-CAN framework

- **Arbitrary identifiers** - As time is divided into ECs, and scheduled in slots according to EDF, the CAN message identifiers do not play a *significant* role in the message response-time. With native CAN the message identifier determines the response-time of a message. Using Server-CAN, however, a message is guaranteed to be transmitted within the EC in which it has been scheduled by the M-Server for transmission. The message identifier merely identifies the contents of a message frame, and decides when within an EC a message will be sent. This is very good from a subsystem integration point of view.

- **Arbitrary scheduler** - As the scheduling of the network is performed at the M-Server, any scheduling algorithm can be used. The Server-CAN framework allows for server-based scheduling algorithms relying on the EDF scheduling algorithm, that provides optimality in terms of schedulability, in contrast with the FPS scheduling algorithm implemented by the native CAN MAC protocol.

- **Combination of traffic** - The usage of the Server-CAN framework provides bandwidth isolation among N-Servers. Since all messages are sent through N-Servers, event-triggered traffic will not interfere with time-triggered traffic if they are separated using different N-Servers. Hence, event- and time-triggered traffic can be combined using the same basic principles, the N-Servers, not disturbing each other as would be the case when using native CAN. The N-Servers acts as software bus guardians, preventing babbling idiots from disturbing the network traffic.

- **Bandwidth reclaiming** - By using a STOP-message as an indicator of the end of an EC, slack is collected from messages not being sent, messages being shorter (in bytes) than originally intended, and messages being shorter due to the bit-stuffing mechanism [147]. This slack is efficiently reused by the immediate start of the following EC, hence providing better performance compared with non slack reclaiming schedulers (see Chapter 8).

- **Advanced protocol mechanisms** - The usage of the Server-CAN framework allows for the implementation of several advanced protocol mechanisms:
  - **Admission control** - Supporting open systems [41], the Server-CAN admission control implemented in the M-Server allows users
Summary

of the network to dynamically add, remove and update N-Servers during runtime, providing controlled communications flexibility compared with native CAN.

- **Mode changes** - Changing from one operational mode to another where the usage of the network might completely change is required by several applications. This can easily be handled thanks to the mode changing functionality provided by the M-Server.

- **Bandwidth sharing** - As the Server-CAN framework provides bandwidth isolation among N-Servers and as the centralised M-Server is continuously monitoring (polling) the network, bandwidth sharing algorithms [18, 120] can be implemented. This provides dynamic adaptation and flexibility to the scheduling of the network during run-time, allowing for efficient use of the sparse bandwidth offered by CAN.

However, running the Server-CAN framework does not only provide benefits. Also, a number of weak points are introduced that have to be considered and evaluated against the strong points when investigating the use of Server-CAN for a particular application.

- **Single point of failure** - Introducing a centralised scheduler running on a special node in the system is a single point of failure. The M-Server schedules all message traffic on the network. Hence, failure of the M-Server could cause a severe failure of the whole system. A solution to this is to have replicated M-Servers in the system. These replicated M-Servers are called backup M-Servers (B-Servers). However, replication of M-Servers introduces another problem: maintaining the consistency among the replicated M-Servers.

- **Protocol overhead** - The Server-CAN protocol introduces overhead in terms of computing power required to run the Server-CAN software responsible for maintaining the M-Server scheduling of the N-Servers (as well as maintaining the B-Servers). Also, some computing power is required to run the N-Servers at all nodes in the system. However, the N-Servers are very simple and do not consume much resources.

On the network, overhead is introduced in terms of protocol messages that are continuously being sent (the TM and STOP messages). The amount of overhead caused by protocol messages depends on the length
10.4 General applicability

As the research presented in the thesis is tightly coupled to the CAN network, it is important to discuss the general applicability of the work, i.e., is there a place for the research presented in this thesis outside the context of CAN.

From a general point of view, the Server-CAN scheduling framework is an implementation of a centralised framework for polling-based share-driven scheduling algorithms, and the framework does not rely on CAN at all. What make Server-CAN efficient is the characteristics of CAN in terms of its message collision resolution mechanism. The MAC protocol of CAN implements CSMA/CR which makes it possible to reclaim unused bandwidth in an efficient way. Another network technology with similar functionality is Byteflight
(as well as one part of FlexRay), which implements FTDMA. Also here, unused bandwidth can be reclaimed, although not as efficient as for CAN. For Byteflight and FlexRay a minislotting mechanism is used that reclaim unused bandwidth. Hence, there is a potential to implement a scheduling framework similar to the Server-CAN framework on these network technologies.

The work in this thesis is a centralised polling-based scheduler (a master) that relies on the possibility to mediate a schedule to its slaves together with the possibility to poll how the schedule is utilised. This polling is needed by the centralised scheduler to update its scheduling parameters. To get high efficiency, it should be possible to reclaim unused bandwidth (slack) in some way. This is very hard for, e.g., TDMA based network technologies, but easier when using, e.g., CSMA/CR or CSMA/CA based network technologies.

In general, the algorithms implemented by the Server-CAN framework are useful in applications that must be polled. For example, distributed systems that require some kind of system wide decision must mediate the states of the nodes participating in the decision to the decision making algorithm. Here it might be desirable for the decision making algorithm to poll the nodes. For example, an application can have strict requirements on its resource usage, i.e., in resource constrained systems an objective is to minimise the number of messages sent over the network. This is what is done using the approach in this thesis, i.e., the N-Servers are polled due to the low bandwidth offered by the CAN network.

10.5 Conclusions

The Server-CAN framework proposed in this thesis provides a more efficient and flexible integration of subsystems compared to the current approaches using standard CAN or TDMA based networks.

Integration of subsystems is done today using standard CAN, sometimes with assisting tools such as the Volcano concept. However, these approaches do not provide very flexible run-time behaviour as the usage of the message identifiers is specified pre-run-time.

Using a TDMA implementation of CAN also separate the message identifiers from the message priority. Hence, a TDMA implementation of CAN allows for predictable integration of subsystems. However, such an implementation is not very flexible both during the development of the system and especially during run-time of the system.

What is unique with the Server-CAN framework is that it (1) provides an
efficient and predictable scheduling of the messages, (2) decouples the message identifier from the priority allowing for predictable integration of subsystems, and (3) allows for a flexible run-time configuration where the system parameters can change over time.

10.6 Future work

The future work includes:

- Implementation and evaluation of the Server-CAN framework in a real system
  The theoretical specification and analysis of the Server-CAN framework is given in this thesis. The next step is to implement the Server-CAN framework in a real system.

- Investigate the overhead of the M-Server (the scheduler)
  Once the Server-CAN framework is implemented in a real system, it is possible to evaluate the overhead of using a centralised M-Server. In fact, the M-Server is doing quite much work polling the network and scheduling the ECs.

- Fault-tolerance of the Server-CAN framework
  In order to use Server-CAN in safety-critical applications its fault tolerant operation also has to be analysed. In Chapter 6 the fault model intended to be used for the fault-tolerant operation of Server-CAN has been presented. Multiple combinations of channel transient faults can occur and their impact on the Server-CAN protocol has to be analysed in detail. Protocol messages can be both omitted and duplicated in either a consistent or inconsistent way.

  Babbling applications are tolerated by Server-CAN as long as they are caused by software faults. Note that some node physical faults may cause a global failure. In particular, the M-Server is a single point of failure and hence M-Server replication using B-Servers is required in order to tolerate physical faults. B-Servers require consistency and transient channel faults are an impairment to this requirement. Suitable mechanisms must be incorporated in order to guarantee consistency of M- and B-Servers under faults and this work has not yet finished.
A possible advantage of Server-CAN is its inherent ability in improving the fault tolerance capabilities of the system, by having a flexible method for taking care of channel faults, as they can be scheduled as aperiodic messages without having the need to reserve bandwidth a priori. Together with replicated M-Servers, the Server-CAN approach could provide a fault-tolerant solution for applications where dependability is a primary requirement.
Appendix A

List of publications (full)

A Published material that forms the basis of this thesis


Abstract: This paper investigates the concept of share-driven scheduling of networks with real-time properties. Share-driven scheduling provides fairness and bandwidth isolation between predictable as well as unpredictable streams of messages on the network.

The need for this kind of scheduled real-time communication network is high in applications that have requirements on flexibility, both during development for assigning communication bandwidth to different applications, and during run-time to facilitate dynamic addition and removal of system components.

We illustrate the share-driven scheduling concept by applying it to the popular Controller Area Network (CAN), resulting in a new share-driven scheduler called Server-CAN. By separating streams of messages using network access servers (N-Servers), scheduling is performed at three layers, where native CAN message arbitration is the scheduling at the lowest level. On top of this is the server-based scheduling to separate different streams of messages within the system. Looking at a single N-Server, several users might share a single server. Hence, scheduling is performed at the third level every time the N-Server is being scheduled.
for message transmission. Here different queuing policies play a role in the scheduling performed.

**Contributors:** Thomas Nolte is the main contributor and main author of the paper, and the work has been carried out in close cooperation with Mikael Nolin and Hans Hansson.

**Usage in thesis:** This paper contributes to material on Server-CAN presented in Chapter 6, Chapter 7 and Chapter 8.


**Abstract:** This paper presents a state-of-practice (SOP) overview of automotive communication technologies, including the latest technology developments. These networking technologies are classified in four major groups: (1) *current wired*, (2) *multimedia*, (3) *upcoming wired* and (4) *wireless*. Within these groups a few technologies stand out as strong candidates for future automotive networks. The goal of this paper is to give an overview of automotive applications relying on communications, identify the key networking technologies used in various automotive applications, present their properties and attributes, and indicate future challenges in the area of automotive communications.

**Contributors:** Thomas Nolte is the main contributor and main author of the paper. The paper was initiated by Thomas while he was visiting Lucia Lo Bello at University of Catania in Italy during 2004.

**Usage in thesis:** This paper contributes to the material on automotive communications presented in Chapter 4.

Abstract: This work-in-progress (WIP) paper presents Server-CAN and highlights its operation and possible vulnerabilities from a fault-tolerance point of view. The paper extends earlier work on Server-CAN by investigating the behaviour of Server-CAN in faulty conditions. Different types of faults are described, and their impact on Server-CAN is discussed.

Contributors: Thomas Nolte is the main author of the paper. The work is carried out in close cooperation with Julian Proenza and Guillermo Rodriguez-Navas from Universitat de les Illes Balears in Spain, who provides extensive knowledge in the area of fault-tolerant CAN, as well as with Sasikumar Punnekkat contributing with knowledge of dependability. The paper was initiated by Thomas Nolte and Guillermo Rodriguez-Navas during a bus ride in Sicily 2004.

Usage in thesis: This paper contributes to a section on fault-tolerant operation of Server-CAN in Chapter 6.


Abstract: This paper investigates the concept of share-driven scheduling, using servers, of networks with real-time properties. Share-driven scheduling provides fairness and bandwidth isolation between predictable as well as unpredictable streams of messages on the network.

The need for this kind of scheduled real-time communication network is high in applications that have requirements on flexibility, both during development for assigning communication bandwidth to different applications, and during run-time to facilitate dynamic addition and removal of system components.

The share-driven scheduling concept is illustrated by applying it to the popular Controller Area Network (CAN), and a scheduling mechanism is proposed called Simple Server-Scheduled CAN (S3-CAN), for which an associated timing analysis is presented. Additionally, a variant of S3-CAN called Periodic Server-Scheduled CAN (PS2-CAN) is presented, which for some network configurations gives lower worst-case response-times than S3-CAN. Also for this improvement, a timing analysis is presented. Simulation is used to evaluate the timing performance
of both S³-CAN and PS²-CAN, comparing them with other scheduling mechanisms.

**Contributors:** Thomas Nolte is the main contributor and main author of the paper, and the work has been carried out in close cooperation with Mikael Nolin and Hans Hansson. The is an invited paper, based on our paper published at ETFA in Lisbon 2003.

**Usage in thesis:** This paper is the core paper of the thesis, contributing to material on Server-CAN presented in Chapter 6, Chapter 7 and Chapter 8.


**Abstract:** This book chapter is presenting concepts and issues related to real-time in embedded systems.

**Contributors:** Hans Hansson, Mikael Nolin and Thomas Nolte are co-authors of the book chapter, with Thomas Nolte being responsible for the sections on real-time scheduling and real-time communication. The work was initiated by Hans Hansson during 2003.

**Usage in thesis:** This book chapter contributes to material presented in Chapter 2 and Chapter 4.


**Abstract:** In this paper, we discuss the impact of the evolution of embedded systems on the design and usage of Real-Time Operating Systems (RTOS). Specifically, we consider issues that result from the integration of complex requirements for embedded systems. Integration
has been identified as a complex issue in various fields such as automotive, critical systems (aerospace, nuclear etc) and consumer electronics. In addition, the pressure on time-to-market, the emergence of multi-site development, and the ever-increasing size of software stacks are driving radical changes in the development approaches of modern applications. These complex requirements have placed greater requirements on Operating Systems with respect to how interfaces are defined and how resources are managed. These requirements are expanded and justified through the course of this paper. The requirements are then discussed in the context of emerging solutions from a number of domains.

Contributors: Thomas Nolte is a coauthor of this joint paper, responsible for emerging issues and the parts on automotive systems. The work has been carried out online with contributions by all authors. The paper was initiated by Thomas Nolte and Clara M. Otero Perez as a follow up of the initiation of a similar paper by Iain Bate at the ERTSI workshop in Lisbon 2004. During winter 2005 all authors finalised this paper for OSPERT.

Usage in thesis: This paper contributes to the motivation of this thesis, presented in Chapter 1.


Abstract: This paper presents an overview of wireless automotive communication technologies, with the aim of identifying the strong candidates for future in-vehicle and inter-vehicle automotive applications. The paper first gives an overview of automotive applications relying on wireless communications, with particular focus on telematics. Then, the paper identifies the key networking technologies used for in-vehicle and inter-vehicle applications, comparing their properties and indicating future challenges in the area of wireless automotive communications, with a focus on real-time aspects.

Contributors: Thomas Nolte is the main contributor and main author of the paper. The paper was initiated by Thomas Nolte while he was visiting Lucia Lo Bello at University of Catania in Italy during 2004,
and finalised together with Lucia Lo Bello and Hans Hansson during spring 2005.

Usage in thesis: This paper contributes to the material presented in Chapter 4.


Abstract: This book chapter presents concepts and issues relating to real-time systems.

Contributors: Hans Hansson, Mikael Nolin and Thomas Nolte are co-authors of the book chapter, with Thomas Nolte being responsible for the sections on real-time scheduling and real-time communication. The work was initiated by Hans Hansson during 2003.

Usage in thesis: This book chapter contributes to material presented in Chapter 2 and Chapter 4.


Abstract: In-car electronics plays an important role in many automotive applications, such as, steer-by-wire and brake-by-wire, and is expected to gradually replace mechanical or hydraulic means to control these systems. The number of electronic components in a car has therefore significantly grown, thus leading to important implications for the vehicle engineering process. In particular, in-car networks used to interconnect electronics equipments are a key point. While in the past extensive use of wiring was a common design practice, nowadays, for the sake of reducing the vehicle weight and fuel consumption, in-car bus networks are largely adopted.

This paper points out current automotive communication standards, i.e., CAN, LIN, Byteflight and MOST, together with upcoming automotive communication standards, namely, TT-CAN, TTP and FlexRay. The work focuses on discussing open issues with the current FlexRay implementation.
Contributors: Thomas Nolte is the main contributor and main author of the paper. The paper was initiated by Thomas Nolte and written while he was visiting Lucia Lo Bello at University of Catania in Italy during 2004.

Usage in thesis: This paper contributes to the material presented in Chapter 4.


Abstract: Server-based scheduling of CAN has been proposed as a way of fair scheduling of the Controller Area Network (CAN). By separating streams of messages using network access servers (N-Servers), scheduling is performed at three layers where native CAN message arbitration is the scheduling at the lowest level. On top of this is the server-based scheduling to separate different streams of messages within the system. Looking at a single N-Server, several streams might share one server. Hence, scheduling is performed at the third level every time the N-Server is being scheduled for message transmission. Here different queuing policies play a role in the scheduling performed.

This paper discusses the hierarchical scheduling of CAN, as a way of fair separation of message streams while providing a flexible core mechanism, the server-based scheduling of CAN.

Contributors: Thomas Nolte is the main contributor and main author of the paper, and the work has been carried out in close cooperation with Mikael Nolin and Hans Hansson. The paper was initiated by Thomas Nolte during the INCOM conference in Salvador da Bahia 2004.

Usage in thesis: This paper contributes to a section on hierarchical scheduling in Chapter 6.

Abstract: This paper presents a share-driven scheduling protocol for networks with real-time properties. The protocol provides fairness and bandwidth isolation among predictable as well as unpredictable streams of messages on the network. The need for this kind of scheduled real-time communication network is high in applications that have requirements on flexibility, both during development for assigning communication bandwidth to different applications, and during run-time to facilitate dynamic addition and removal of system components.

The share-driven scheduling protocol is illustrated by applying it to the popular Controller Area Network (CAN). Two versions of the protocol are presented together with their associated timing analysis.

Contributors: Thomas Nolte is the main contributor and main author of the paper, and the work has been carried out in close cooperation with Mikael Nolin and Hans Hansson. The paper was initiated by Thomas Nolte during the ETFA conference in Lisbon 2003.

Usage in thesis: This paper contributes to material presented in Chapter 6, and forms the basis for the material presented in Chapter 7.


Abstract: In this paper we present a new share-driven server-based method for scheduling messages sent over the Controller Area Network (CAN). Share-driven methods are useful in many applications, since they provide both fairness and bandwidth isolation among the users of the resource. Our method is the first share-driven scheduling method proposed for CAN. Our server-based scheduling is based on Earliest Deadline First (EDF), which allows higher utilization of the network than using CAN’s native fixed-priority scheduling approach.

We use simulation to show the performance and properties of server-based scheduling for CAN. The simulation results show that the bandwidth isolation property is kept, and they show that our method provides a Quality-of-Service (QoS), where virtually all messages are delivered within a specified time.

Contributors: Thomas Nolte is the main contributor and main author of the paper, and the work has been carried out in close cooperation with
Mikael Nolin and Hans Hansson. The paper was initiated by Thomas Nolte as a result of the research conducted at University of California, Irvine, CA, USA during Thomas’ stay with professor Kwei-Jay Lin 2002.

Usage in thesis: This paper contributes to material presented in Chapter 6.


Abstract: We present a new share-driven server-based method for scheduling messages sent over the Controller Area Network (CAN). Share-driven methods are useful in many applications, since they provide both fairness and bandwidth isolation among the users of the resource. Our method is the first share-driven scheduling method proposed for CAN, and it is based on Earliest Deadline First (EDF), which allows higher utilization of the network than CAN’s native fixed-priority scheduling (FPS). We use simulation to show the performance and properties of server-based scheduling for CAN. The simulation results show that the bandwidth isolation property is kept, and that with our method a good Quality-of-Service (QoS) is provided, where virtually all messages are delivered within their deadline.

Contributors: Thomas Nolte is the main contributor and main author of the paper, and the work has been carried out in close cooperation with Mikael Nolin and Hans Hansson. The paper was initiated by Thomas Nolte as a result of the research conducted at University of California, Irvine, CA, USA during Thomas’ stay with professor Kwei-Jay Lin 2002.

Usage in thesis: This paper contributes to material presented in Chapter 6.

Abstract: The Controller Area Network (CAN) is a widely used real-time communication network for automotive and other embedded applications. As new applications continue to evolve, the complexity of distributed CAN based systems increase. However, CAN’s maximum speed of 1 Mbps remains fixed, leading to performance bottlenecks. In order to make full use of this scarce bandwidth, methods for increasing the achievable utilisation are needed.

Traditionally, real-time scheduling theory has targeted hard real-time systems, which most of the time are safety critical. Since these systems (by definition) are not allowed to have any timing flaws, analysis techniques need to take all possible scenarios of execution combinations and execution times of the system into consideration. This will result in a system that is configured for the worst possible scenario. Whether this scenario is likely, or even possible, in the real system is not considered. Hence, the result may be an unnecessarily expensive system, with potentially overly provisioned resources.

In this thesis we address two issues. In the first part, we investigate how to loosen up pessimistic real-time analysis in a controlled way, thereby allowing the designer to make well-founded trade-offs between the level of real-time guarantee and the system cost. Specifically, we investigate and model the bit-stuffing mechanism in CAN in order to retrieve representative distributions of stuff-bits, which we then use in the response time analysis instead of the worst-case values normally used. We evaluate the validity of these stuff-bit distributions in two case studies, and we integrate this representation of message frame length with the classical CAN worst-case response-time analysis.

In the second part of the thesis, we propose a novel way of scheduling the CAN. By providing server-based scheduling, bandwidth isolation between users is guaranteed. This increases the flexibility of CAN, by providing efficient handling of sporadic and aperiodic message streams. Server-based scheduling also has the potential to allow higher network utilisation compared to CAN’s native scheduling. The performance and properties of server-based scheduling of CAN is evaluated using simulation. Also, the server-based scheduling is applied in an end-to-end analysis.

Contributors: Thomas Nolte is the main contributor and author of this thesis. The work was performed under supervision of Hans Hansson, Christer Norström and Sasikumar Punnekkat.
Usage in thesis: This paper contributes to material presented in Chapter 2, Chapter 5 and Chapter 6.


Abstract: Distributed real-time applications share a group of processors connected by some local area network. A rigorous and sound methodology to design real-time systems from independently designed distributed real-time applications is needed. In this paper, we study a distributed real-time system design scheme using CBS-based end-to-end scheduling. The scheduling scheme utilizes CBS to allocate both CPU shares and network bandwidth to a distributed real-time application when it arrives at the system. Our proposed solution uses the same scheduling paradigm for both resources. In this way, we believe the system can have a more consistent scheduling objective and may achieve a tighter schedulability condition.

Contributors: This paper is based on an idea of Kwei-Jay Lin. Thomas Nolte is responsible for the CAN and server parts, and Kwei-Jay Lin for the rest.

Usage in thesis: This paper is the original paper of share-driven scheduling for CAN. The paper contributes to material presented in Chapter 6.

B Additional journal publications (not included in thesis)


C Additional conference publications (not included in thesis)


D Additional workshop publications (not included in thesis)


23 Approaches to Support Real-Time Traffic over Bluetooth Networks, Lucia Lo Bello (University of Catania, Italy), Mario Collotta (University of Catania, Italy), Orazio Mirabella (University of Catania, Italy), Thomas Nolte, In Proceedings of the 4th International Workshop on Real-Time Networks (RTN’05) in conjunction with the 17th Euromicro International Conference on Real-Time Systems (ECRTS’05), pages 47-50, Palma de Mallorca, Balearic Islands, Spain, ISBN 3-929757-90-7, Editor: Jörg Kaiser, July 2005.


E Other publications (not included in thesis)


Appendix B

List of acronyms

ABS  Antilock Braking System
ACC  Adaptive Cruise Control
ACK  ACKnowledgement
AM   Asynchronous Message
ARINC Aeronautical Radio, Inc.
AUTOSAR AUTomotive Open System ARchitecture
B-Server Backup M-Server
BC   Basic Cycle
BSS  Bandwidth Sharing Server
BWI  BandWidth Inheritance protocol
CAL  CAN Application Layer
CAN  Controller Area Network
CBS  Constant Bandwidth Server
CBS-R Constant Bandwidth Server with Resource constraints
CC   Cruise Control
CCK  Complementary Code Keying
CD   Compact Disc
CEM  Central Electronic Module
CiA  CAN in Automation
CPU  Central Processing Unit
CRC  Cyclic Redundancy Check
CSMA Carrier Sense Multiple Access
CSMA/CA Carrier Sense Multiple Access / Collision Avoidance
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSMA/CD</td>
<td>Carrier Sense Multiple Access / Collision Detection</td>
</tr>
<tr>
<td>CSMA/CR</td>
<td>Carrier Sense Multiple Access / Collision Resolution</td>
</tr>
<tr>
<td>D2B</td>
<td>Domestic Digital Bus</td>
</tr>
<tr>
<td>DCCS</td>
<td>Distributed Computer Control System</td>
</tr>
<tr>
<td>DEcos</td>
<td>Dependable Embedded Components and Systems</td>
</tr>
<tr>
<td>DLC</td>
<td>Data Length Code</td>
</tr>
<tr>
<td>DPC</td>
<td>Dynamic Priority Ceiling</td>
</tr>
<tr>
<td>DPE</td>
<td>Dynamic Priority Exchange</td>
</tr>
<tr>
<td>DPI</td>
<td>Dynamic Priority Inheritance</td>
</tr>
<tr>
<td>DPS</td>
<td>Dynamic Priority Scheduling (Scheduler)</td>
</tr>
<tr>
<td>DS</td>
<td>Deferrable Server</td>
</tr>
<tr>
<td>DS-UWB</td>
<td>Direct Sequence UWB</td>
</tr>
<tr>
<td>DSC</td>
<td>Dynamic Stability Control</td>
</tr>
<tr>
<td>DSI</td>
<td>Distributed Systems Interface</td>
</tr>
<tr>
<td>DSS</td>
<td>Dynamic Sporadic Server</td>
</tr>
<tr>
<td>DSSS</td>
<td>Direct Sequence Spread Spectrum</td>
</tr>
<tr>
<td>DVD</td>
<td>Digital Versatile Disc</td>
</tr>
<tr>
<td>EAS</td>
<td>Electronic Automotive System</td>
</tr>
<tr>
<td>EC</td>
<td>Elementary Cycle</td>
</tr>
<tr>
<td>ECU</td>
<td>Electronic Control Unit</td>
</tr>
<tr>
<td>EDC</td>
<td>Electronic Damper Control</td>
</tr>
<tr>
<td>EDF</td>
<td>Earliest Deadline First</td>
</tr>
<tr>
<td>EDL</td>
<td>Earliest Deadline Late</td>
</tr>
<tr>
<td>EMI</td>
<td>Electro Magnetic Interference</td>
</tr>
<tr>
<td>EOF</td>
<td>End Of Frame</td>
</tr>
<tr>
<td>ESP</td>
<td>Electronic Stability Program</td>
</tr>
<tr>
<td>F-CAN</td>
<td>BMW version of CAN</td>
</tr>
<tr>
<td>FHSS</td>
<td>Frequency Hopping Spread Spectrum</td>
</tr>
<tr>
<td>FIFO</td>
<td>First In First Out</td>
</tr>
<tr>
<td>FPS</td>
<td>Fixed Priority Scheduling (Scheduler)</td>
</tr>
<tr>
<td>FTDMA</td>
<td>Flexible TDMA</td>
</tr>
<tr>
<td>FTT</td>
<td>Flexible Time-Triggered</td>
</tr>
<tr>
<td>FTT-CAN</td>
<td>Flexible Time-Triggered CAN</td>
</tr>
<tr>
<td>FTT-Ethernet</td>
<td>Flexible Time-Triggered Ethernet</td>
</tr>
<tr>
<td>GPS</td>
<td>General Processor Sharing</td>
</tr>
<tr>
<td>GPS</td>
<td>Global Positioning System</td>
</tr>
<tr>
<td>GSM</td>
<td>Global System for Mobile communications</td>
</tr>
<tr>
<td>IDE</td>
<td>IDentifier Extension</td>
</tr>
<tr>
<td>IEC</td>
<td>International Engineering Consortium</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>IEEE</td>
<td>Institute of Electrical and Electronics Engineers</td>
</tr>
<tr>
<td>INT</td>
<td>INTer-frame space</td>
</tr>
<tr>
<td>IPE</td>
<td>Improved Priority Exchange</td>
</tr>
<tr>
<td>ISO</td>
<td>International Organization for Standardization</td>
</tr>
<tr>
<td>K-CAN</td>
<td>BMW version of CAN</td>
</tr>
<tr>
<td>LAN</td>
<td>Local Area Network</td>
</tr>
<tr>
<td>LIN</td>
<td>Local Interconnect Network</td>
</tr>
<tr>
<td>LLF</td>
<td>Least Laxity First</td>
</tr>
<tr>
<td>LoCAN</td>
<td>BMW version of CAN</td>
</tr>
<tr>
<td>LST</td>
<td>Least Slack Time first</td>
</tr>
<tr>
<td>M-Server</td>
<td>Master Server</td>
</tr>
<tr>
<td>MAC</td>
<td>Medium Access Control</td>
</tr>
<tr>
<td>MI</td>
<td>Motorola Interconnect</td>
</tr>
<tr>
<td>MIL</td>
<td>Military</td>
</tr>
<tr>
<td>MML</td>
<td>Mobile Multimedia Link</td>
</tr>
<tr>
<td>MOST</td>
<td>Media Oriented Systems Transport</td>
</tr>
<tr>
<td>MP3</td>
<td>Moving Picture experts group layer-3 audio</td>
</tr>
<tr>
<td>MTS</td>
<td>Mixed Traffic Scheduler</td>
</tr>
<tr>
<td>MVB</td>
<td>Multifunction Vehicle Bus</td>
</tr>
<tr>
<td>N-Server</td>
<td>Network access Server</td>
</tr>
<tr>
<td>NMEA</td>
<td>National Marine Electronics Association</td>
</tr>
<tr>
<td>NoW</td>
<td>Network on Wheels</td>
</tr>
<tr>
<td>OBD</td>
<td>On Board Diagnostics</td>
</tr>
<tr>
<td>ODVA</td>
<td>Open DeviceNet Vendor Association</td>
</tr>
<tr>
<td>OFDM</td>
<td>Orthogonal Frequency Division Multiplexing</td>
</tr>
<tr>
<td>OSI</td>
<td>Open Systems Interconnection</td>
</tr>
<tr>
<td>P2P</td>
<td>Point to Point</td>
</tr>
<tr>
<td>PAN</td>
<td>Personal Area Network</td>
</tr>
<tr>
<td>PCP</td>
<td>Priority Ceiling Protocol</td>
</tr>
<tr>
<td>PDC</td>
<td>Producer-Distributor-Consumer(s)</td>
</tr>
<tr>
<td>PE</td>
<td>Priority Exchange</td>
</tr>
<tr>
<td>PIP</td>
<td>Priority Inheritance Protocol</td>
</tr>
<tr>
<td>PP-CAN</td>
<td>Periodic Polling scheduled CAN</td>
</tr>
<tr>
<td>PROFIBUS</td>
<td>Process Field Bus</td>
</tr>
<tr>
<td>PS</td>
<td>Polling Server</td>
</tr>
<tr>
<td>PS²-CAN</td>
<td>Periodic Server-Scheduled CAN</td>
</tr>
<tr>
<td>PSS-CAN</td>
<td>Periodic Server-Scheduled CAN</td>
</tr>
<tr>
<td>PT-CAN</td>
<td>BMW version of CAN</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality of Service</td>
</tr>
<tr>
<td>Acronym</td>
<td>Definition</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
</tr>
<tr>
<td>RATAD</td>
<td>Reliability And Timing Analysis of Distributed systems</td>
</tr>
<tr>
<td>RM</td>
<td>Rate Monotonic</td>
</tr>
<tr>
<td>RM</td>
<td>Reference Message</td>
</tr>
<tr>
<td>RTR</td>
<td>Remote Transmission Request</td>
</tr>
<tr>
<td>S^3-CAN</td>
<td>Simple Server-Scheduled CAN</td>
</tr>
<tr>
<td>SAE</td>
<td>Society of Automotive Engineers</td>
</tr>
<tr>
<td>SAVE</td>
<td>component based design for SAfety critical VEHicular systems</td>
</tr>
<tr>
<td>SCI</td>
<td>Serial Communication Interface</td>
</tr>
<tr>
<td>Server-CAN</td>
<td>Server scheduled CAN</td>
</tr>
<tr>
<td>SI-BUS</td>
<td>BMW version of Byteflight</td>
</tr>
<tr>
<td>SIG</td>
<td>Special Interest Group</td>
</tr>
<tr>
<td>SM</td>
<td>Synchronous Message</td>
</tr>
<tr>
<td>SOF</td>
<td>Start Of Frame</td>
</tr>
<tr>
<td>SOP</td>
<td>State Of Practice</td>
</tr>
<tr>
<td>SRP</td>
<td>Stack Resource Policy</td>
</tr>
<tr>
<td>SS</td>
<td>Sporadic Server</td>
</tr>
<tr>
<td>SSS-CAN</td>
<td>Simple Server-Scheduled CAN</td>
</tr>
<tr>
<td>STOP</td>
<td>STOP message</td>
</tr>
<tr>
<td>TBS</td>
<td>Total Bandwidth Server</td>
</tr>
<tr>
<td>TCN</td>
<td>Train Communication Network</td>
</tr>
<tr>
<td>TCP/IP</td>
<td>Transmission Control Protocol/Internet Protocol</td>
</tr>
<tr>
<td>TDMA</td>
<td>Time Division Multiple Access</td>
</tr>
<tr>
<td>TK</td>
<td>ToKen</td>
</tr>
<tr>
<td>TM</td>
<td>Trigger Message</td>
</tr>
<tr>
<td>TT-CAN</td>
<td>Time-Triggered CAN</td>
</tr>
<tr>
<td>TTA</td>
<td>Time-Triggered Architecture</td>
</tr>
<tr>
<td>TTP</td>
<td>Timed Token Protocol</td>
</tr>
<tr>
<td>TTP</td>
<td>Time-Triggered Protocol</td>
</tr>
<tr>
<td>UART</td>
<td>Universal Asynchronous Receiver-Transmitter</td>
</tr>
<tr>
<td>USB</td>
<td>Universal Serial Bus</td>
</tr>
<tr>
<td>UWB</td>
<td>Ultra Wide Band</td>
</tr>
<tr>
<td>VDC</td>
<td>Vehicle Dynamics Control</td>
</tr>
<tr>
<td>VTCSMA</td>
<td>Virtual Time CSMA</td>
</tr>
<tr>
<td>WCET</td>
<td>Worst-Case Execution Time</td>
</tr>
<tr>
<td>Wi-Fi</td>
<td>Wireless Fidelity</td>
</tr>
<tr>
<td>WLAN</td>
<td>Wireless LAN</td>
</tr>
<tr>
<td>WorldFIP</td>
<td>World Factory Instrumentation Protocol</td>
</tr>
<tr>
<td>WTB</td>
<td>Wire Train Bus</td>
</tr>
</tbody>
</table>
# Appendix C

## List of notations

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\epsilon$</td>
<td>minimum time quantum</td>
</tr>
<tr>
<td>$\Delta$</td>
<td>a small time offset</td>
</tr>
<tr>
<td>$a$</td>
<td>a constant used to vary system load in the experiments</td>
</tr>
<tr>
<td>$b$</td>
<td>a constant used to vary server load in the experiments</td>
</tr>
<tr>
<td>$b$</td>
<td>the shortest period of all N-Servers</td>
</tr>
<tr>
<td>$B$</td>
<td>blocking time</td>
</tr>
<tr>
<td>$c$</td>
<td>remaining capacity</td>
</tr>
<tr>
<td>$C$</td>
<td>capacity</td>
</tr>
<tr>
<td>$\overline{C}$</td>
<td>worst-case transmission/execution time</td>
</tr>
<tr>
<td>$\overline{C}$</td>
<td>message transmission time</td>
</tr>
<tr>
<td>$d$</td>
<td>absolute deadline</td>
</tr>
<tr>
<td>$D$</td>
<td>relative deadline</td>
</tr>
<tr>
<td>$</td>
<td>EC</td>
</tr>
<tr>
<td>$g$</td>
<td>bits exposed to bit-stuffing</td>
</tr>
<tr>
<td>$h_{[t_1,t_2]}$</td>
<td>processor demand in time interval $t \in [t_1, t_2]$</td>
</tr>
<tr>
<td>$h(t)$</td>
<td>processor demand in time interval $t \in [0, t]$</td>
</tr>
<tr>
<td>$h$</td>
<td>number of queued messages</td>
</tr>
<tr>
<td>$hp(i)$</td>
<td>set of messages/tasks with priority higher than that of message/task $i$</td>
</tr>
<tr>
<td>$I$</td>
<td>interference from higher priority tasks</td>
</tr>
<tr>
<td>$J$</td>
<td>jitter</td>
</tr>
<tr>
<td>$k$</td>
<td>number of required N-Servers</td>
</tr>
</tbody>
</table>
lep(i) set of messages/tasks with priority lower than or equal to that of message/task i
lp(i) set of messages/tasks with priority lower than that of message/task i
L number of data bytes in a CAN message frame
M set of message identifiers
m message identifier
n message identifier
M size of a message in bits
modeChangeREQ mode change request
N the set of N-Servers in the system
|N| the number of N-Servers in the system
n number of messages
N the number of tasks in the system
nodeID node identifier
O protocol overhead
P set of protocol messages
P priority
q queuing time
Q bandwidth
Q^{max} maximum bandwidth
Q^{system_load} system load
r release time
R worst-case latency/response-time
R^{backlog} worst-case response-time in backlog mode
R^{normal} worst-case response-time in normal mode
replyCMD reply command
s N-Server ID
S set of streams
S stream
serverID server identifier
STOP size of a STOP message in bits
\tau bit time
t an arbitrary time
\tau_{phs} random initial phase for N-Servers
\tau_{phu} random initial phase for users
T period
T^{min} minimum period
T^{max} maximum period
LIST OF NOTATIONS

\( T \) \hspace{1cm} N-Server period
\( T_s \) \hspace{1cm} period of an N-Server
\( T_s^{\text{blocked}} \) \hspace{1cm} duration of the penalty for an N-Server being blocked by other N-Servers
\( T_s^{\text{not ready}} \) \hspace{1cm} duration of the penalty for an N-Server not being ready once scheduled
\( T_s^{\text{queued}} \) \hspace{1cm} duration of the penalty for an N-Server having to wait for other N-Servers to transmit their messages
\( T_{EC} \) \hspace{1cm} duration of an EC
\( T_M \) \hspace{1cm} duration of a message transmission
\( T_{S^3-CAN}^{\text{pen}} \) \hspace{1cm} duration of the \( S^3\)-CAN penalty
\( T_{sched} \) \hspace{1cm} the computational overhead of updating N-Server absolute deadlines, producing the new schedule and encoding it into a TM
\( T_{STOP} \) \hspace{1cm} duration of a STOP message
\( T_{TM} \) \hspace{1cm} duration of a TM message
\( TM \) \hspace{1cm} size of a TM message in bits
\( \text{updateCMD} \) \hspace{1cm} update command
\( W \) \hspace{1cm} shifted delay
Bibliography


Operating System Platforms for Embedded Real-Time Applications (OSPERT’05) in conjunction with the 17th Euromicro International Conference on Real-Time Systems (ECRTS’05), pages 13–19, Palma de Mallorca, Balearic Islands, Spain, July 2005.


[98] H. Kopetz. TTP/A - A time-triggered protocol for body electronics using standard UARTs. In SAE *World Congress*, pages 1–9, Detroit, MI, USA, 1995. SAE.


[136] Motorola. The MI BUS and product family for multiplexing systems. Motorola publication EB409/D.


[181] SAE J1850 Standard - The Society of Automotive Engineers (SAE) Vehicle Network for Multiplexing and Data Communications Standards
Committee. Class B Data Communications Network Interface, May 1994, SAE.


[183] SAE J1939 Standard - The Society of Automotive Engineers (SAE) Truck and Bus Control and Communications Subcommittee. SAE J1939 Standards Collection, 2004, SAE.


Index

14V electrical power, 54
42V electrical power, 54

ABS, 51
absolute deadline, 37
ACC, 51
accelerator pedal, 54
acknowledgement, 18, 19
actuator, 12, 25
output, 12
Adaptive Cruise Control, see ACC
air condition, see body and comfort electronics
airbag, 11, see passive safety
ALOHA, 18
AM, 99
Antilock Braking System, see ABS
aperiodic, 14
application domain
aerospace, 1
automotive, see automotive domain
avionics, see avionic domain
building and home automation systems, see building and home automation domain
consumer electronics, 1
industrial automation, see industrial and process automation domain
medical, 12
trains, see train domain
application layer, see ISO/OSI reference model
ARINC 429, 82
avionic domain, 82
ARINC 629, 82
avionic domain, 82
asynchronous message, see AM
periodic task, 38
audio system, see multimedia and infotainment automation
network, 26
office, 26
automotive producers, 48
system, see EAS
system vendors, 64
automotive domain, 1, 5, 12, 45–82
Bluetooth, see Bluetooth
Byteflight, see Byteflight
CAN, see CAN
D2B, see D2B
DSI, see DSI
FlexRay, see FlexRay
IDB-1394 (Automotive Firewire), see IDB-1394 (Automotive Firewire)
IEEE 802.11, see IEEE 802.11
IEEE 802.15.1, see IEEE 802.15.1
IEEE 802.15.3a, see IEEE 802.15.3a
IEEE 802.15.4, see IEEE 802.15.4
LIN, see LIN
MI, see MI
MML Bus, see MML Bus
MOST, see MOST
Safe-by-Wire, see Safe-by-Wire
TT-CAN, see TT-CAN
TTP/A, see TTP/A
TTP/C, see TTP/C
UWB, see UWB, see UWB
Wi-Fi, see Wi-Fi
ZigBee, see ZigBee
Automotive Open System Architecture, see AUTOSAR
AUTOSAR, 45
avionic domain, 12, 63, 82–83
ARINC 429, see ARINC 429
ARINC 629, see ARINC 629
Byteflight, see Byteflight
CAN, see CAN
MIL-1553, see MIL-1553
TTP/C, see TTP/C
B-Server, 120–122
characterisation, 120
run-time behaviour, 121
babbling idiot, 49
backoff
algorithm, 20
time, 19
bandwidth, 50, see EAS, communication requirements
BandWidth Inheritance protocol, see BWI
Bandwidth Sharing Server, see BSS
Basic Cycle, see BC
BC, 71
blocking, 13, 34
factor, 41
time, 37, 39
Bluetooth, 76
audio system, 76
Bluetooth SIG, see Bluetooth SIG
Hands-Free profile, 76
IEEE 802.15.1, see Bluetooth mobile phone, 76
QoS, see QoS
voice application, 78
wireless PAN, see wireless PAN
Bluetooth SIG, 76
BMW 7 series, 58
body and comfort electronics, 53
air condition, 53
climate control, 53
Cruise Control, see CC
locks, 53
park distance control, 53
seat control, 53
window lifts, 53
brake-by-wire, see x-by-wire
BSS, 34
building and home automation domain, 63, 84
LonWorks, see LonWorks
X10, see X10
bus, see topology
guardian, 49
topology, 27, see network
busy period
deadline-i, 42
level-i, 41
BWI, 36
Byteflight, 23, 65–67, 82
airbag, 65
avionic domain, 82
clock synchronisation, 66
communication cycle, 66
master node, 66
minislot, 66
plastic optical fibre, 67
seat belt pretensioner, 65
slot counter, 66
star coupler, 66
sync master, 66
sync pulse, 66
CAL, 102
cam timing, 52
CAN, 1, 21, 47, 57, 65, 82, 83, 87–102
applications, 90
arbitration mechanism, 90
avionic domain, 82
body and comfort electronics, 65, 90
CAN Application Layer,
see CAL
CAN in Automation, see CiA
control and diagnostics, 65
CSMA/CR, 90
electrical characteristics, 90
engine control, 90
frame arbitration, 90
frame identifier, 90
higher layer protocols, 97–102
academic, 97–100
CAN Kingdom, see
CAN Kingdom
CANopen, see CANopen
commercial, 101–102
DeviceNet, see DeviceNet
EDF, 97–98
FTT-CAN, see FTT-CAN
Server-CAN, see Server-CAN
TT-CAN, see TT-CAN
history, 87–88
industrial and process auto-
mation domain, 83
ISO 11519-2, 87
ISO 11783, 89
ISO 11898, 87, 88
ISO 11992, 88, 89
machinery for agriculture and
forestry, 89
marine usage, 89
NMEA 2000®, 89
Open DeviceNet Vendor Associ-
ation, see ODVA
SAE J1850, 88
SAE J1939, 88, 89
SAE J2284, 88
schedulability analysis, 92–97
bit stuffing, 93
classical analysis, 92–94
extensions to classical analy-
sis, 95–97
fault models, 96–97
message transmission time,
93–94
probabilistic analysis, 95
reduction of pessimism, 97
response-time analysis, 92–97
scheduling, 91–92
FTT-CAN, 91
priority-driven, 91
Server-CAN, 92
share-driven, 92
time-driven, 91
TT-CAN, 91
standards, 88–89
technical properties, 90–91
topology
bus, 91
star, 91
tractor, 89
train domain, 83
tuck and trailer application, 88
x-by-wire, 65
CAN Antrieb, see VW Passat
CAN Application Layer, see CAL
CAN in Automation, see CiA
CAN Infotainment, see VW Passat
CAN Kingdom, 101
CAN Komfort, see VW Passat
CANopen, 102
car manufacturer, 45, see automotive
car stereo, see multimedia and info-
tainment
Car2Car Consortium, 75, 77
Carrier Sense Multiple Access, see CSMA
Carrier Sense Multiple Access / Collision Avoidance, see CSMA/CA
Carrier Sense Multiple Access / Collision Detection, see CSMA/CD
Carrier Sense Multiple Access / Collision Resolution, see CSMA/CR
carrier sensing, 19
CarTALK 2000, 77
CBS, 33
CBS-R, 36
CC, 53
CCK, 78
CD, 81
ceiling priority, 36
system, 36
CEM, see Volvo XC90
Central Electronic Module, see CEM
chassis, 51–52
Adaptive Cruise Control, see ACC
Antilock Braking System, see ABS
Dynamic Stability Control, see DSC
Electronic Damper Control, see EDC
Electronic Stability Program, see ESP
Vehicle Dynamics Control, see VDC
Chevrolet Corvette, 54
CiA, 90, 102
climate control, see body and comfort electronics
collision avoidance, 21
detection, 20
domain, 26
message, 26
resolution, 21
resolution algorithm, 26
communication protocol, 16
requirement, 45, see EAS
communications adapter, 17
channel, 2
INDEX

in-vehicle, 75
inter-vehicle, 75
network, 16
wired, 16
wireless, 16
real-time, 16
vehicle-to-roadside, 75
vehicle-to-vehicle, 75
Compact Disc, see CD
Complementary Code Keying, see CCK
consortia, 45, see consortium
car2Car, 75
FlexRay, 72
The LIN Consortium, 64
Constant Bandwidth Server, see CBS
Constant Bandwidth Server with
Resource constraints, see CBS-R
control
application, 2
function, 12
system
DCCS, see DCCS
controller
parallel interface, see interface
serial interface, see interface
Controller Area Network, see CAN
cost, 46, 48, see EAS, communication requirements
CPU, 3, 12, 25
Crash
Detection, 52
situation, 52
CRC, 74
critical
instant, 38, 41
section, 34
Cruise Control, see CC
CSMA, 19
CSMA/CA, 17, 21
CSMA/CD, 17, 20
CSMA/CR, 17, 21, 65
Cyclic Redundancy Check, see CRC

D2B, 68
Data Length Code, see DLC
data link layer, see ISO/OSI reference model
DCCS, 12
deadline, 12
absolute, 37
relative, 37
deadline-i busy period, 42
deadlock, 35
DECOS, 71
Deferrable Server, see DS
demand-assignment protocol, see MAC
demand-based test, 37, 39–40
DeviceNet, 102
diagnostics, 45, 56
tool, 50
Digital Versatile Disc, see DVD
Direct Sequence Spread Spectrum, see DSSS
Direct-Sequence UWB, see DS-UWB
dispatcher, 28
distributed
architecture, 3
computer control system, see DCCS
computer system, 45
embedded architecture, 45
memory, 12
system, 3, 12
Distributed Computer Control Systems, see DCCS
Distributed Systems Interface, see DSI
DLC, 94
Domestic Digital Bus, see D2B
DPC, 36
DPE, 33
DPI, 36
DPS, 30
driving axis, 52
DS, 32
DS-UWB, 78
DSC, 51
DSI, 68
DSS, 33
DSSS, 78
DVD, 53, 81
player, see multimedia and infotainment
dynamic priority, see DPS
scheduler, see DPS
Dynamic Priority Ceiling, see DPC
Dynamic Priority Exchange, see DPE
Dynamic Priority Inheritance, see DPI
dynamic schedule, 4
Dynamic Sporadic Server, see DSS
Dynamic Stability Control, see DSC

Earliest Deadline First, see EDF
Earliest Deadline Late, see EDL
EAS, 1, 5, 45–62

body and comfort electronics, see body and comfort electronics
chassis, see chassis
communication requirements, 48–50
general, 48–49
technical, 49–50
current trends, 48
diagnostics, 56
example
BMW 7 series, see BMW 7 series
Volvo XC90, see Volvo XC90
VW Passat, see VW Passat
historical perspective, 47
multimedia and infotainment, see multimedia and infotainment
passive safety, see passive safety
powertrain, see powertrain
wireless and telematics, see wireless
x-by-wire, see x-by-wire

EC, 99, 100, 115
ECU, 45, 57
EDC, 51
EDF, 30
EDL, 33
electric
actuator, 45
sensor, 45
Electro Magnetic Interference, see EMI
electromagnetic
actuator, 45
interference, see EMI
sensor, 45
Electronic Automotive System, see EAS
Electronic Control Unit, see ECU
Electronic Damper Control, see EDC
electronic gear control, see powertrain
Electronic Stability Program, see ESP
Elementary Cycle, see EC
embedded application, 2
computers, 47
network, 63–85
system, 1
EMI, 129
End Of Frame, see EOF
game, 52
control, see powertrain
speed, 52
EOF, 94
error containment, 49
ESP, 51
Ethernet, 20, 25–27, 83
industrial and process
automation domain, 83
event
input, 12
task, 12
time, 12
event-triggered, 1, 3, 50, 65, 67,
71–73
exact behaviour, 30
execution
time, 12
worst-case, 16
F-CAN, see BMW 7 series
factory communication network, 25
fault tolerance, 45, 46, 49,
see EAS, communication
requirements
fault tolerant
communication
defective circuits, 49
line failures, 49
operation, 45
feasible, 37
FHSS, 78
fieldbus, 25, 47
FIFO, 107, 110, 113
First In First Out, see FIFO
fixed priority, see FPS
scheduler, see FPS
fixed-assignment protocol, see MAC
fixed-point iteration, 41
flexibility, 50, see EAS, communication
defective circuits, 49
fault tolerant
Feasible, 37
FHSS, 78
fieldbus, 25, 47
FIFO, 107, 110, 113
First In First Out, see FIFO
fixed priority, see FPS
scheduler, see FPS
fixed-assignment protocol, see MAC
fixed-point iteration, 41
flexibility, 50, see EAS, communication
Flexible TDMA, see FTDMA
Flexible Time-Triggered framework, see FTT
FlexRay, 23, 45, 72–74
bus guardian, 73
bus topology, 73
clock synchronisation, 73
communication cycle, 73
CRC, see CRC
cycle trigger, 74
dual channel, 73
dynamic segment, 73
error containment, 74
error detection, 74
event-triggered mode, 73
fault tolerant, 74
master, 74
minislot, 73
safety-critical, 74
single channel, 73
star topology, 73
static segment, 73
static slot, 73
symbol window, 74
time-triggered mode, 73
x-by-wire, 72, 74, 80
force feedback, 54
FPS, 30
fraction, *see* share
Frequency Hopping Spread Spectrum, *see* FHSS
FTDMA, 18, 23, 66, 73
FTT, 26
FTT-CAN, 91, 99
   Elementary Cycle, *see* EC
   Trigger Message, *see* TM
FTT-Ethernet, 26
fuel injection, 52
future proof, 48, *see* EAS, communication requirements
gear-box, 52
gear-by-wire, *see* shift-by-wire
   General Processor Sharing, *see*
   GPS (General Processor Sharing)
Global Positioning System, *see*
   GPS (Global Positioning System)
Global System for Mobile communications, *see* GSM
global window, 26
GPS (General Processor Sharing), 31
GPS (Global Positioning System), 53, 55, 67
GPS and navigation system, *see*
   multimedia and infotainment
GSM, 22
hard real-time, 15
hierarchical scheduler, 33
hydraulic automotive subsystem
   braking, 45
   steering, 45
IDB-1394 (Automotive Firewire), 68
IDE, 94
IDentifier Extension, *see* IDE
IEC 61275, 83, *see* TCN
IEEE 802.11, 77, *see* Wi-Fi
IEEE 802.15.1, 76, *see* Bluetooth
IEEE 802.15.3a, 77, *see* UWB
IEEE 802.15.4, 76, *see* ZigBee
IEEE 802.5 Token Ring Protocol, 24, 26
Improved Priority Exchange, *see* IPE
in-vehicle communications, 75
industrial and process automation domain, 12, 63, 83–84
   CAN, *see* CAN
   Ethernet, *see* Ethernet
   INTERBUS, *see* INTERBUS
   PROFIBUS, *see* PROFIBUS
   PROFInet, *see* PROFInet
   TTP/C, *see* TTP/C
   WorldFIP, *see* WorldFIP
   ZigBee, *see* ZigBee
Industrial Ethernet, 83
   train domain, 83
input event, 12
INT, 94
integration, *see* subsystem integration
INTer-frame space, *see* INT
inter-vehicle communications, 75
INTERBUS, 83
   industrial and process automation domain, 83
interface
   parallel, 25
serial, 25
internet connectivity, see multimedia and infotainment
IPE, 33
ISO 11519-2, 87, see CAN
ISO 11783, 89, see CAN
ISO 11898, 65, 87, see CAN, 88
ISO 11898-4, see TT-CAN
ISO 11992, 88, see CAN, 89
ISO/OSI protocol stack, 17
ISO/OSI reference model, 16
application layer, 16
data link layer, 16
network layer, 16
physical layer, 16
presentation layer, 16
session layer, 16
transport layer, 16
K-CAN, see BMW 7 series
kernel
real-time, 34
LAN, 77
laxity, 31
lead time, 48, see EAS, communication requirements
Least Laxity First, see LLF
Least Slack Time first, see LST
legacy
hardware, 1
software, 1
level-i busy period, 41
LIN, 24, 57, 64
body and comfort electronics, 64
TTP/A comparison, 69
LIN/CAN gateway, 64
LLF, 31
Local Area Network, see LAN
Local Interconnect Network, see LIN
LoCAN, see BMW 7 series
locks, see body and comfort electronics
LonWorks, 84
building and home automation domain, 84
LST, 31
M-Server, 113–120
parameters, 115
Elementary Cycle, see EC
set of N-Servers, 115
protocol, 115–118
Stop message, see STOP
Trigger Message, see TM
run-time behaviour, 120
run-time mechanisms, 118–120
run-time monitoring, 119–120
scheduler, 118–119
support for advanced run-time mechanisms, 120
MAC, 16–25
protocol, 16–25
demand-assignment, 17
fixed-assignment, 17
random access, 17
master, 18, see M-Server
node, 24, see M-Server
master/slave, 24, 26, 64
Media Oriented Systems Transport, see MOST
medium, 25
cable, 25
electrical, 25
optical, 25
optical fibre, 25
radio, 25
Medium Access Control, see MAC
Mercedes Benz, 48, 68, 90
F400, 48
message, 2
  collision, 26
duplication
  consistent, 129
  inconsistent, 129
inter-arrival time, 2
omission
  consistent, 129
  inconsistent, 129
periodic, 2
stream, 2
transmission, 2
MI, 68
microprocessor, 25
middleware, 63
MIL-1553, 82
  avionic domain, 82
mini slotting, 23, 66
Mixed Traffic Scheduler, see MTS
MML Bus, 68
Mobile Multimedia Link, see MML Bus
mode change, 127
model
  real-time, 12–16
task, 37
monitor and display, see multimedia
  and infotainment
MOST, 57, 67
  active speaker, 67
  automotive multimedia
    applications, 67
digital radio, 67
GPS navigation, 67
multimedia and infotainment, 67
video display, 67
motor, 54
Motorola Interconnect, see MI
Moving Picture experts group layer-3
  audio, see MP3
MP3, 81
MTS, 98
Multifunction Vehicle Bus, see MVB
multimedia and infotainment, 53
  audio system, 53
car stereo, 53
  DVD player, 53
GPS and navigation system, 53
internet connectivity, 53
monitor and display, 53
MOST, see MOST
speaker, 53
video game, 53
multiprocessor system, 12
mutual exclusion, 34
MVB, 83
N-Server, 110–113
  parameters, 111–112
    absolute deadline, 111
    bandwidth, 111
    capacity, 111
    identity, 111
    period, 111
    relative deadline, 111
    remaining capacity, 111
    set of associated messages, 112
    user, 111
  run-time behaviour, 113
  run-time mechanisms, 112–113
    message queue, 112
    scheduler, 113
network
  automation, 26
  communications, 16
Ethernet, 25
factory communication, 25
fieldbus, 25
layer, see ISO/OSI reference model
office, 26
technology, 46
requirement, 45
topology, 27
bus, 27
ring, 27
star, 27
wireless, 27
network layer, see ISO/OSI reference model
network technology, 63
automotive domain
Bluetooth, see Bluetooth
Byteflight, see Byteflight
CAN, see CAN
current wired, 63–69
D2B, see D2B
discussion, 79
DSI, see DSI
FlexRay, see FlexRay
IDB-1394 (Automotive Firewire), see IDB-1394 (Automotive Firewire)
IEEE 802.11, see IEEE 802.11
IEEE 802.15.1, see IEEE 802.15.1
IEEE 802.15.3a, see IEEE 802.15.3a
IEEE 802.15.4, see IEEE 802.15.4
LIN, see LIN
MI, see MI
MML Bus, see MML Bus
MOST, see MOST
Safe-by-Wire, see Safe-by-Wire
TT-CAN, see TT-CAN
TTP/A, see TTP/A
TTP/C, see TTP/C
upcoming wired, 63, 69–74
UWB, see UWB, see UWB
Wi-Fi, see Wi-Fi
wireless, 63, 75–79
ZigBee, see ZigBee
avionic domain, 82–83
ARINC 429, see ARINC 429
ARINC 629, see ARINC 629
Byteflight, see Byteflight
CAN, see CAN
MIL-1553, see MIL-1553
TTP/C, see TTP/C
building and home automation domain, 84
LonWorks, see LonWorks
X10, see X10
industrial and process automation domain, 83–84
CAN, see CAN
Ethernet, see Ethernet
INTERBUS, see INTERBUS
PROFIBUS, see PROFIBUS
PROFINet, see PROFINet
TTP/C, see TTP/C
WorldFIP, see WorldFIP
ZigBee, see ZigBee
train domain, 83
CAN, see CAN
Industrial Ethernet, see Industrial Ethernet
TCN, see TCN
TTP/C, see TTP/C
WorldFIP, see WorldFIP
Network-on-Wheels, see NoW
NMEA 2000®, 89, see CAN
non pre-emptive, 13
   EDF scheduling, 40
      fixed-priority, 41
NoW, 77
OBD, 56
ODVA, 102
OFDM, 78
office
   automation, 26
      network, 26
offline, 22, 28
      scheduler, 28, see scheduler
      scheduling, 26
On Board Diagnostics, see OBD
online, 22, 28
      scheduler, 28, see scheduler
Open DeviceNet Vendor Association, see ODVA
optical, 25
   fibre, 25
Orthogonal Frequency Division Multiplexing, see OFDM
OSI reference model, see ISO/OSI
   reference model
output actuator, 12
P2P, 78
PAN, 76
parallel interface, see interface
      park distance control, see body and
      comfort electronics
partitioning
   spatial, 3
      temporal, 3
passive safety, 52
   airbag, 52
   seat belt pretensioner, 52
PCP, 36
PDC, 83
PE, 32
period, 12, 37
   periodic, 14
Periodic Polling scheduler,
   see PP-CAN
Personal Area Network, see PAN
Personal Mobility vehicle, 48
physical layer, see ISO/OSI
   reference model
PIP, 35, 36
plastic optical fibre
   MOST, 67
plug-and-play, 68
Point to Point, see P2P
polling, 64
Polling Server, see PS
powertrain, 52
   electronic gear control, 52
      engine control, 52
PP-CAN, 162–164
   timing analysis, 163
      updating N-Server states,
      162–163
pre-emption, 13
   level, 36
pre-emptive, 13
precedence relation, 11
predictability, 45, 49, see EAS, communication requirements
presentation layer, see ISO/OSI
   reference model
priority, 13, 37
   assignment, 30
      ceiling, 36
      inversion, 34
Priority Ceiling Protocol, see PCP
Priority Exchange, see PE
Priority Inheritance Protocol, see PIP
priority-driven, see scheduling mechanism
     scheduler, 28, see scheduler
problem description, 3–5
processor system
     multiprocessor system, 12
     single processor system, 12
Producer-Distributor-Consumer, see PDC
PROFIBUS, 24, 83
     industrial and process automation domain, 83
PROFIBUS - DP, 84
PROFIBUS - FMS, 84
PROFIBUS - PA, 84
PROFInet, 83
     industrial and process automation domain, 83
program
     real-time, 12
proposition, 5, 216
PS, 31
PS²-CAN, 137–142
     M-Server scheduling, 138–140
timing analysis, 140–142
     updating N-Server states, 137–138
PT-CAN, see BMW 7 series
pyrotechnic device, 52
QoS, 76, 100
Quality of Service, see QoS
radar, 51
radio, 25
random access protocol, see MAC
RATAD, 5
Rate Monotonic, see RM (Rate Monotonic)
real-time
     analysis, 11, 37–42
     communications, 11, 16–28
constraint
     hard, 15
     soft, 15
     violation, 15
kernel, 34
program, 12
requirement
     hard, 15
     soft, 15
     violation, 15
scheduler, 28
     scheduling, 11, 28–36
     system, 11–12
     non pre-emptive, 13
     pre-emptive, 13
system model, 12
     system modelling, 12–16
recurrence relation, 41
relative deadline, 37
release time, 37
     Remote Transmission Request, see RTR
research
     contributions, 7–8, 214–221
method, 6
project
     RATAD, see RATAD
     SAVE, see SAVE
questions, 5–6
     revisited, 214–216
work, 6
response-time, 12, 37
test, 37, 40–42
ring, see topology
ring topology, 27, see network
RM (Rate Monotonic), 30
RM (Reference Message), 101, 106
RTR, 94
SAE J1850, 65, 88, see CAN
SAE J1939, 65, 88, see CAN, 89
SAE J2284, 65, 88, see CAN
Safe-by-Wire, 68
safety-critical, 45
system, 69
SAVE, 5
scenario
worst-case, 11
schedulability, 30
test, 37
schedule
dynamic, 4
static, 2, 4
scheduler
dynamic priority, see DPS
fixed priority, see FPS
hierarchical, 33–34
offline, 28
online, 28
priority-driven, 28, 30–31
real-time, 28
share-driven, 28, 31–33
time-driven, 28–29
scheduling, see scheduler
scheduling mechanism, see scheduler
seat
belt, 52
belt pretensioner, see passive safety
control, see body and comfort electronics
security, 50, see EAS, communication requirements
semaphore, 34
signal, 34
wait, 34
sensor, 12, 25
acceleration, 51
brake pressure, 51
humidity, 53
pinion angle, 54
rotary movement, 51
sonar distance, 53
speed, 51
steering wheel angle, 51, 54
temperature, 53
serial bus, 47
serial interface, see interface
server, 31
bandwidth, 31
capacity, 31
period, 31
priority, 31
server-based
scheduler, 31
Server-CAN, 1, 5, 92, 100, 103–131
advanced run-time mechanisms, 124–127
admission protocol, 125–126
bandwidth sharing, 127
mode change, see mode change
concept, 107–108
Elementary Cycle, see EC
evaluation, 165–196
PS²-CAN, 176
S³-CAN, 179–188
comparison, 188
PP-CAN, 170–174
simulation result format, 169
simulation set up, 166–169
fault tolerance, 127–130
channel permanent faults, 129
channel transient faults, 129
consistency, 130
fault model, 127–129
node physical faults, 128–129
node software faults, 128
framework, 1, 103–131, 216–219
general applicability, 219–220
master, see M-Server
overhead, 116–118
protocol basics, 109–122
B-Server, see B-Server
M-Server, see M-Server
N-Server, see N-Server
scheduler, 133–161
PS²-CAN, see PS²-CAN
S³-CAN, see S³-CAN
general properties, 134–137
PP-CAN, see PP-CAN
scheduler design, 104–106
state
B-Server state, 120
M-Server state, 120
Trigger Message, see TM
usage, 197–212
configuration, 198–205
cost, 208–212
example, 205–212
response-time analysis, 202
subsystem integration, 203–205
service methods, 31
session layer, see ISO/OSI reference model
share, 3, 31
share-driven, see scheduling mechanism
framework, see Server-CAN
scheduler, 28, 31
scheduling framework, see Server-CAN
shared resource, 34
shift lever, 54
shift-by-wire, see x-by-wire
shock absorber, 51
SI-BUS, see BMW 7 series
SIG, 76
signal, 25
single processor system, 12
slack, 31
Slack Stealer, 32
slave, 24
slot, 22, 23
SM, 99
SOF, 94
soft real-time, 15
spatial partitioning, 3
speaker, see multimedia and infotainment
Special Interest Group, see SIG
sporadic, 14
Sporadic Server, see SS
SRP, 36
SS, 32
S³-CAN, 142–161
example, 161
M-Server scheduling, 144–145
timing analysis, 145–161
backlog mode, 154–160
normal mode, 146–154
worst case, 160–161
updating N-Server states, 143–144
Stack Resource Policy, see SRP
star, see topology
star topology, 27, see network
Start Of Frame, see SOF
static schedule, 2, 4
steer-by-wire, see x-by-wire
STOP, 115–116
stream, 107, 110, 111
sub layer, 16
subsystem, see EAS
subsystem integration, 1–5, 60, 197–198, 203–205, 213–214
switch, 26
synchronous
message, see SM
periodic task, 38
system
architecture, 1
celling, 36
DCCS, 12
distributed, 3, 12
embedded, 1
model
real-time, see real-time
real-time, 1, see real-time
table, 29
task, 12
aperiodic, 14
arrival, 14
event, 12
inter-arrival time, 14
model, 37
periodic, 14
property, 12
deadline, 12
period, 12
priority, 13
WCET, 12
real-time
constraint, see real-time
requirement, see real-time
requirement, 12
set, 12, 30
sporadic, 14
utilisation, 12
TBS, 33
TCN, 83
train domain, 83
TCP/IP, 16
TDMA, 18, 22, 26, 69, 71, 73, 101
telematics, 50, 55, see wireless
temporal
behaviour, 3
partitioning, 3
performance, 1, see real-time
requirements, 1
test
demand-based test, see demand-based test
response-time test, see response-time
schedulability test, 37
utilisation-based, see utilisation-based test
throttle-by-wire, see x-by-wire
time
blocking, 37, 39
event, 12
execution, 12
instant, 4
partition, 2
release, 37
response, 37
response time, see response-time
slot, 2
WCET, see WCET
Time Division Multiple Access, see TDMA
time to market, see lead time
time-driven, see scheduling mechanism
scheduler, 28, see scheduler
time-triggered, 50, 64, 67, 71–73
Time-Triggered Architecture, see TTA
Time-Triggered CAN, see TT-CAN
Time-Triggered Protocol, see TTP
Timed Token Protocol (TTP), 24, 26
timeliness, 11
timing, 11
   analysis, see real-time
   constraints, see real-time
guarantees, 2
TK, see token
TM, 99, 100, 106, 115
token, 18, 24
Token algorithm, 26
topology, 27
   network, 27
   bus, 27
   ring, 27
   star, 27
Total Bandwidth Server, see TBS
Toyota, 48
trade-off
   cost and bandwidth, 50
   performance and cost, 46
traffic
   asynchronous, 67
   synchronous, 67
traffic smoothing, 27
Train Communication Network, see TCN
train domain, 12, 63, 83
CAN, see CAN
Industrial Ethernet, see Industrial Ethernet
TCN, see TCN
TTP/C, see TTP/C
WorldFIP, see WorldFIP
transmission, 54
Transmission Control Protocol/Internet Protocol, see TCP/IP
transport layer, see ISO/OSI reference model
Trigger Message, see TM
TT-CAN, 71–72, 91, 101
   basic cycle, 71
   clock synchronisation, 71
   cycle counter, 72
   Level 1, 72
   Level 2, 72
Reference Message (RM), see
   Reference Message (RM)
system matrix, 72
time window
   arbitration window, 71
   exclusive window, 71
   free window, 71
   reference message, 71
time-master, 71
TTA, 69
   DECOS, see DECOS
TTP, 69–71
   avionics, 80
   x-by-wire, 80
TTP/A, 24, 69
   LIN comparison, 69
TTP/C, 22, 69, 82, 83
   avionic domain, 71, 82
   fault tolerance, 71
   fault tolerant mechanism, 70
   distributed clock synchronisation, 70
   membership service, 70
   industrial and process automation domain, 83
optical medium, 71
replicated channels, 71
safety-critical, 70
TDMA round, 70
train domain, 83
twisted pair, 71

UART/SCI, 64
Ultra Wide Band, see UWB
Universal Asynchronous Receiver-
Transmitter/Serial Communication Interface,
see UART/SCI
Universal Serial Bus, see USB
USB, 69
user, 107, 110, 111
utilisation
Server-CAN, 116–118, 208–212
task, 12
utilisation-based test, 37–39
UWB, 77
collision-detection system, 77
IEEE 802.15.3a, see UWB
multimedia device, 77
 suspension system, 77

valve control, 52
VDC, 51
vehicle active safety system, see chassis
Vehicle Dynamics Control, see VDC
vehicle passive safety systems, see passive safety
vehicle-to-roadside communications, 75
vehicle-to-vehicle communications, 75
video game, see multimedia and infotainment

Virtual Time CSMA, see VTCSMA
Volcano concept, 60
Volvo XC90, 57
VTCSMA, 26
VW Passat, 59–60

WCET, 12, 16, 37
Wi-Fi, 77
IEEE 802.11, see Wi-Fi
window lifts, see body and comfort electronics
window protocol, 26
Wire Train Bus, see WTB
wired, 16
wireless, 16

communication networks, see network
connection, 50
connectivity, 55
network, 27
telematics, 55
wireless and telematics, 55–56
advanced driver assistance, 55
car access systems, 55
cell phones, 55
diagnostics and maintenance services, 56
fleet management systems, 55
GPS units, 55
laptop computers, 55
navigation and traffic information systems, 55
safety systems, 55
security systems, 56
voice, 56

wireless and telematics, see wireless
Wireless Local Area Network, see WLAN
wireless PAN, 76
INDEX

WLAN, 77
WorldFIP, 83
    industrial and process automation domain, 83
    train domain, 83
worst-case
    response-time, see response-time scenario, 11
Worst-Case Execution Time,
    see WCET
WTB, 83

x-by-wire, 54–55
    non safety-critical, 54
    shift-by-wire, 54
    throttle-by-wire, 54
    safety-critical, 54
    brake-by-wire, 54
    steer-by-wire, 54
X10, 84
    building and home automation domain, 84

ZigBee, 21, 76, 83, 84
    air conditioning, 77
    air-conditioning, 82
    building and home automation domain, 84
    heating, 77
    IEEE 802.15.4, see ZigBee
    industrial and process automation domain, 83
    lighting control, 77, 82
    monitoring and control application, 76
    sensor network, 77
    sensors and control, 78
    telemetry, 82
    tire tracking/monitoring, 82
    toll collection, 82
    vehicle identification, 82
    vehicle immobilizers, 82
    ventilation, 77
    ZigBee Alliance, 76
It has been an amazing time of hard work, sometimes combined with rewarding travel. Antibes Juan-les-Pins, London, Los Angeles, Tokyo, San José, Taipei, Washington, DC, Porto, Aveiro, Lisbon, Cancún, Salvador da Bahía, Catania, Piazza Armerina, Palma de Mallorca, Vienna, Miami Beach... Whenever I have the chance, I try to stay a few days extra to enjoy the local culture. The picture on the cover of this thesis (printed version only) was taken during one of these trips where I stayed a few days extra. It is easy to see why...

Don’t forget to enjoy life! :-}