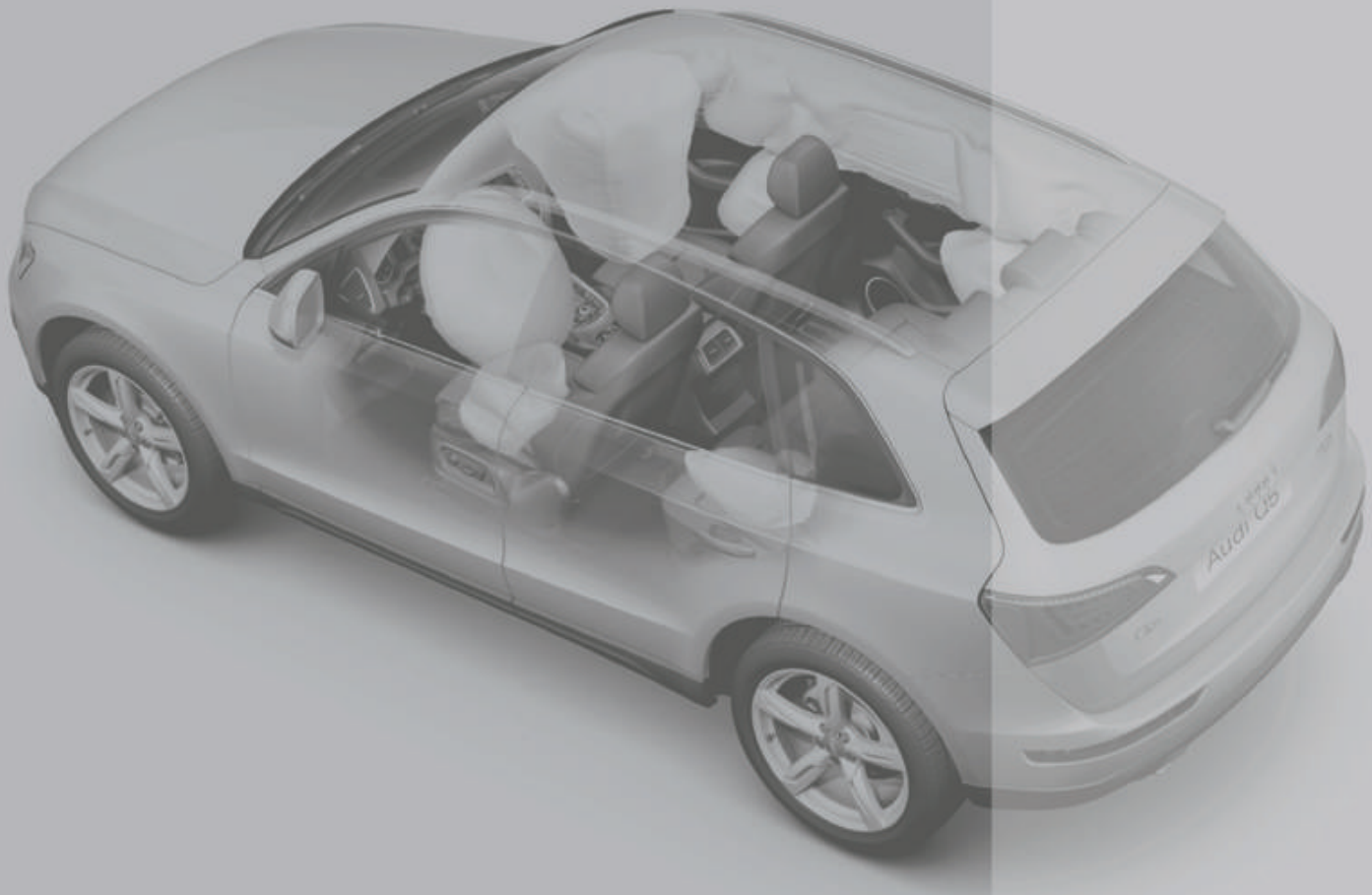
Audi
Dissertationsreihe

Leistungsbewertung zeitkritischer Datenübertragung in automobilen Kommunikationssystemen

Thomas Herpel



Cuvillier Verlag Göttingen

INI.FAU
INGOLSTADT INSTITUTE DER FRIEDRICH-ALEXANDER-UNIVERSITÄT ERLANGEN-NÜRNBERG

Performance Evaluation of Time-Critical Data Transmission in Automotive Communication Systems

Leistungsbewertung zeitkritischer Datenübertragung in automobilen Kommunikationssystemen

Der Technischen Fakultät der
Universität Erlangen-Nürnberg
zur Erlangung des Grades

DOKTOR-INGENIEUR

vorgelegt von

Thomas Herpel

Erlangen - 2009

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.
1. Aufl. - Göttingen : Cuvillier, 2010
Zugl.: Erlangen-Nürnberg, Univ. Diss., 2009

978-3-86955-348-1

Audi Dissertationsreihe, Band 32

Als Dissertation genehmigt von
der Technischen Fakultät der
Universität Erlangen-Nürnberg

Tag der Einreichung:	24.08.2009
Tag der Promotion:	26.11.2009
Dekan:	Prof. Dr.-Ing. Reinhard German
Berichterstatter:	Prof. Dr.-Ing. Reinhard German
	Prof. Dr.-Ing. Jürgen Teich

Nonnenstieg 8, 37075 Göttingen
Telefon: 0551-54724-0
Telefax: 0551-54724-21
www.cuvillier.de

1. Auflage, 2010
Gedruckt auf säurefreiem Papier

978-3-86955-348-1

Contents

List of Figures

List of Tables

Acknowledgements

From the *Chair of Computer Science 7 at the University of Erlangen-Nürnberg*, I would first of all like to thank my doctoral advisor Prof. Dr. Reinhard German for giving me this great chance to do a doctorate in a fascinating and challenging industry project and for the support and appreciation throughout the last 3 years. Another big thank-you goes to my colleagues Dr. Kai-Steffen Hielscher and Dr. Ulrich Klehmet for the productive work on Network Calculus, the cooperativeness and support and the pleasant atmosphere in our working group!

At the *Audi AG*, I was perfectly integrated into the *Department of Safety Electronics*, for what I owe a debt of gratitude to my advisors Steffen Fey and Johannes Salzberger, the supervisors Quirin Sterner and Alexander Pesch and the divisional head Torsten Gollewski. Thank you also for the fruitful discussions, the support and the great interest whenever I came up with new ideas or results! In addition, I would like to thank Dr. Uwe Koser for keeping the Audi research projects running.

Furthermore, I would like to thank my colleague Christoph Sommer for the good and productive collaboration in our office at the University. I am also indebted to Christoph Lauer and Bernhard Kloiber, whose diploma theses remarkably contributed to my research efforts.

Last but not least, I wish to express my deepest gratefulness to my girlfriend Stephanie, my parents Klaus and Christine and to my grandma Olga for their support, motivation and patience all through these exciting and often stressful times of my dissertation!

Abstract

For almost 30 years, the airbag has been part of a car's passive safety components to protect occupants in a crash. In 1980, in the former S-Class from Mercedes-Benz, the system triggered only one driver airbag, though it was composed of 170 parts. Today, highly integrated electronic control units for deployment of front, side or window airbags are standard equipment even in compact cars. At all times, the airbag control was built as an autarkic system for self-contained detection of crash severity and deployment of protection means. With continuing improvements in sensors, electronics and algorithmic concepts for sensor data processing in the airbag control, the time spans between impact and airbag deployment could be reduced to about 30 milliseconds. Every millisecond in reaction time which can be saved, directly contributes to a reduction of injury severity for the occupants. Thus, even shorter time spans for deployment would be desirable. However, the performance of passive safety systems, which *react* on a collision, and of constructive means, in terms of increased stiffness of the vehicle body, is widely exhausted. The networking and interplay of passive and active safety components, like the electronic stability program or the radar-based headway control, appears to be a promising instrument to compensate for this. Active safety components engage *actively* into the vehicle dynamics in case of a dangerous driving situation in order to avoid a crash, e.g. if the car turns into a side-slip or the distance to a vehicle in front falls below a certain threshold. If sensor data, which are computed by active safety components in a dangerous driving situation, are communicated to the airbag control, it will be able to react faster and more effective in case a crash cannot be avoided in the following. For example, previous knowledge of a risky situation might be used to lower algorithmic thresholds for airbag deployment in order to achieve a shorter reaction time in case of emergency compared to a purely passive reaction on the impact. Such a networking of active and passive safety has to be performed using the in-car communication infrastructure, consisting of bus systems like CAN or FlexRay and gateways to interconnect single bus segments. Real-time capabilities and

timeliness of data transfer – which means data arrives at the receiver within a predefined time interval – are of essential importance for the effectiveness of networked applications for occupant protection. Otherwise, the crash might occur before the passive components were able to receive and process sensor data from active safety systems. For development of networked, highly time-critical vehicle safety systems, sophisticated evaluation of data transmission is indispensable at an early stage of system design.

In this dissertation, concepts and methods are presented to conduct comprehensive performance evaluation studies of in-car communication and to support efficient safety systems in future vehicles. A hardware- and software-based measurement infrastructure was built, which allows to figure out effects of data transmission in normal operation mode, the so-called *use-case*, by evaluation of recorded communication data and to derive important system parameters from the samples. These parameters serve as realistic input for a system model, which resembles the functional and timing behavior of relevant system components using UML-based statecharts and discrete event simulation. The simulation allows for capturing various communication scenarios and for obtaining meaningful performance measures, like end-to-end transmission delays. For the field of time-critical safety applications, measures for the *worst-case* of system operation are also vitally important. Therefore, the analytical method of *Network Calculus* was extended to meet the requirements of automotive communication systems and applied to this area for the first time ever. The analytic results resemble guaranteed upper bounds for delays in data transmission and allow, together with the outcomes from real-life measurements and from use-case simulation, for a distinct evaluation of the networked system in early design phases. Performance measures could be obtained for both the use-case and the worst-case of system operation, supporting a comprehensive evaluation, which can be adapted to the actual criticality of the application and leads to a significant increase in performance of future networked vehicle safety systems.

Zusammenfassung

Seit fast 30 Jahren bietet der Airbag als Komponente der passiven Fahrzeugsicherheit den Insassen Schutz bei einem Unfall. 1980, in der damaligen Mercedes-Benz S-Klasse, aktivierte das System lediglich einen Fahrerairbag und bestand aus 170 Bauteilen. Heute sind hochintegrierte, elektronische Steuerungen für Front-, Seiten- oder Kopf-Airbags bereits in Kleinwagen Serienausstattung. Die Airbagsteuerung war seit jeher als autarkes System ausgelegt, das eigenständig die Schwere der Kollision detektiert und die Schutzmittel aktiviert. Durch stetige Verbesserungen im Bereich der Sensorik, der Elektronik im Steuergerät und der algorithmischen Konzepte der Sensordatenverarbeitung konnten die Reaktionszeiten bis zur Auslösung der Airbags auf ca. 30 Millisekunden reduziert werden. Da jede Millisekunde, die die Systeme der passiven Sicherheit nach Kollisionsbeginn früher aktiviert werden können, dazu beiträgt, die Verletzungsschwere der Insassen zu reduzieren, wären jedoch noch kürzere Auslösezeiten wünschenswert.
Passive Konzepte allein, also die *Reaktion* auf eine Kollision, scheinen weitestgehend ausgereizt, ebenso wie konstruktive Maßnahmen zur Erhöhung der Karosseriesteifigkeit. Ein vielversprechender Ansatz ist die Vernetzung von Systemen der passiven Fahrzeugsicherheit mit aktiven Sicherheitskompontenten, wie dem Elektronischen Stabilitätsprogramm oder der Radar-basierten Abstandsregelung. Diese Systeme greifen durch gezielte *Aktionen* in einer Gefahrensituation in das Fahrgeschehen ein um einen Unfall zu verhindern, z.B. wenn das Fahrzeug ins Schleudern gerät oder der Abstand zum vorausfahrenden Fahrzeug einen kritischen Wert unterschreitet. Werden die Sensordaten, die die aktiven Systeme in einer frühen Gefährdungsphase errechnen, an die Airbagsteuerung kommuniziert und es kommt zur Kollision, kann auf diese schneller und effektiver reagiert werden, z.B. indem Auslöseschwellen in den Airbagalgorithmen bei akuter Gefährdung herabgesetzt werden und bei konkretem Eintritt der Kollision die Airbags mit diesem Vorwissen früher gezündet werden können als bei einer rein passiven Systemauslegung. Eine solche Vernetzung von aktiven und passiven Komponenten muss über die Fahrzeug-interne Kommunikationsinfrastruktur aus Bussystemen

wie CAN oder FlexRay und Gateways zur Verbindung einzelner Bussegmente geschehen. Hierbei ist die Echtzeitfähigkeit, also die Übertragung der Daten in einem prädizierbaren Zeitintervall, von essentieller Bedeutung für die Effektivität der vernetzten Fahrzeugsicherheitsfunktionen. Zu lange Übertragungszeiten können dazu führen, dass die Kollision bereits eingetreten ist, bevor die Sensordaten in den passiven Systemen überhaupt empfangen und verarbeitet werden konnten. Für die Entwicklung vernetzter, hochgradig zeitkritischer Systeme der Fahrzeugsicherheit ist somit eine Evaluation der Datenübertragung bereits in frühen Phasen des Entwurfsprozesses unerlässlich.
Die vorliegende Dissertation stellt Konzepte und Methoden vor, die für eine umfassende Leistungsbewertung Fahrzeug-interner Kommunikation erarbeitet wurden und die Auslegung effizienter, zukünftiger Sicherheitssysteme entscheidend unterstützen. Es wurde eine Hardware- und Software-basierte Messinfrastruktur geschaffen, die es erlaubt, anhand aufgezeichneter Kommunikationsdaten Effekte bei der Datenübertragung im realen Fahrzeugbetrieb, dem sogenannten *use-case*, aufzuzeigen und wichtige Systemparameter zu identifizieren. Diese Parameter dienen als realistische Eingabedaten für ein Systemmodell, das Funktionalität und zeitliches Verhalten der relevanten Komponenten mittels UML-basierter Zustandsdiagramme und diskreter Ereignissimulation nachbildet. In der Simulation können verschiedenste Kommunikationsszenarien untersucht und aussagekräftige Leistungskenngrößen, wie Ende-zu-Ende Übertragungslatenzen, ermittelt werden. Gerade im Bereich zeit- und sicherheitskritischer Anwendungen des Insassenschutzes sind jedoch auch Aussagen über das Systemverhalten im schlimmstmöglichen Fall, dem *worst-case*, von großer Relevanz. Um auch für dieses Szenario gültige Ergebnisse zu erlangen, wurde die analytische Methode des *Network Calculus* für Fragestellungen der Fahrzeug-internen Kommunikation erweitert und erstmals angewandt. Die analytischen Resultate stellen garantierte obere Schranken für die Verzögerungen bei der Datenübertragung dar und erlauben, zusammen mit den Erkenntnissen aus den Messungen an realen Fahrzeugen und den Ergebnissen der Simulation, eine frühzeitige Bewertung des vernetzten Sicherheitssystems. Leistungskenngrößen können sowohl für den Normalbetrieb wie auch für schlimmstmögliche Systemszenarien ermittelt werden, wodurch eine umfassende, an die Kritikalität der jeweiligen Anwendung angepasste Bewertung in einer frühen Designphase und eine signifikante Effizienzsteigerung zukünftiger, vernetzter Fahrzeugsicherheitsfunktionen möglich ist.

1 Introduction

This chapter introduces to the field of automotive vehicle safety, safety electronics and concepts for occupant protection. In addition, the motivation for networking of active and passive safety electronics in future vehicle safety scenarios is given. More details on the topics mentioned in the following can be found in [65], [48] and [33].

1.1 Motivation

The most undesirable situation in road traffic is a collision with either other road users or obstacles on or off the road. Thus, development of adequate protection means for occupants in such a severe situation has been started over 50 years ago. In 1952, the first patent for an airbag-like protection system was assigned to American engineers. However, early approaches for inflatable cushions suffered from extensive inflation times, heavy weight, large installation spaces and poor functional safety of the components. As shown in figure 1.1, the whole plot of a collision lasts for only about 150 milliseconds, which is not much more than the time for a blink of an eye, from the first contact until the damaged car comes to standstill again. These timing demands could never be met by inflation strategies based on high-pressure gas, which is why developers switched to composite gas generators as known from military and space rockets in the early 1970's.
It took until 1980 – almost 30 years of research! – to put airbags into serial operation, when Mercedes-Benz offered for the S-Class (W126) a system composed of a driver-airbag and belt tensioners, available as extra equipment for 1526 DM. Nowadays, even compact cars may be equipped with front-, side-, window- and knee-airbags and hence offer a high level of passive occupant protection. Consequently, the number of road traffic fatalities decreased significantly in the last decades since the development of effective occupant protection systems has been

started. This is mainly the merit of distinctively adjusted passive safety systems, consisting of a stiff vehicle body to absorb as much energy in a crash as possible and quickly deployed airbags and pyrotechnical belt tensioners. Of course, legal enforcements during this time, like speed limits or penalties for driving without fastened seat belts, contributed to the reduction of deaths in traffic, too.

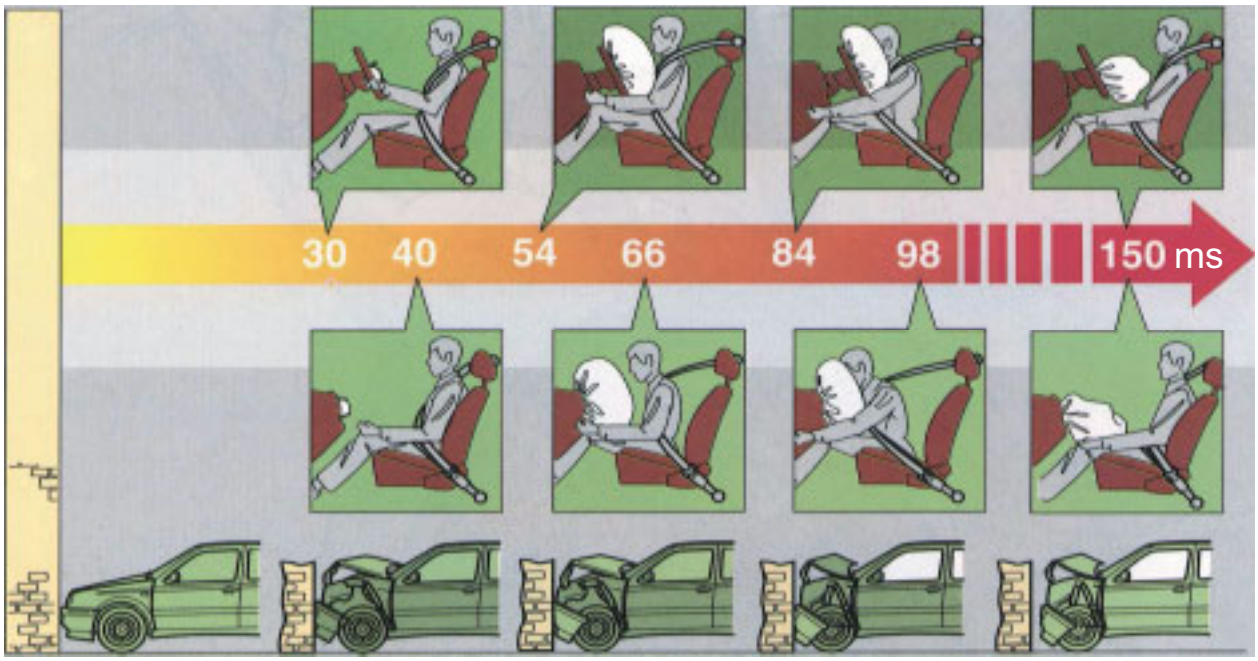


Figure 1.1: Time Plot of Airbag Activation in a Crash [3]

1.2 Airbag Control Systems

1.2.1 Airbag Control Unit

The core component of up-to-date airbag control systems is the airbag control unit (ACU) as shown in figure 1.2. Its main functions are detection of a crash (or a roll-over), activation of adequate protection means, disconnection of the battery from the on-board voltage supply infrastructure and, eventually, transmission of an emergency call in the so-called *post-crash phase* shortly after the collision.

The major electronic components inside the ACU are a microcontroller (*main-*μC) with operating system, deployment algorithms, diagnosis routines etc., and a redundant, second microcontroller (*safety-*μC) for sensor signal processing

Figure 1.2: ACU (left) and Crash Sensors (right: g-Sat, p-Sat) [3]

and deployment decision. Internal acceleration sensors yield measurements on vehicle dynamics, which are used to validate external crash sensor information. To assure autarkic operation of the ACU, e.g. if the battery is damaged early during a severe front-crash, capacitors offer the necessary voltage supply, typically charged to enable 100 to 200 milliseconds of self-sustained operation. Activation of pyrotechnical devices is triggered by a current, that is generated by redundant firing stages inside the ACU. The connection to the in-car communication system is realized by a bus controller. Table 1.1 presents some typical settings for electronics of an airbag control unit. The values are real-life parameters for the ACU of an Audi A6 Limousine, as produced from 2004 to 2007 [4], [3].
Figure 1.3 gives an impression of the typical installation place of the ACU, which is on the gear-tunnel, near to the center of mass in the middle of the car. This placement is chosen for two main reasons. First of all, in case of a crash, the ACU must not be destroyed or disconnected from the contact before belt tensioners and airbags have been deployed. A centered installation of the ACU offers the best protection in terms of distance to possible sites of impact. Second, for reasons of functional safety, the severity of the impact and the acceleration of the vehicle is measured both by external satellite sensors and by internal sensors in the ACU. Installation near to the center of mass, thus near to the central rotation axes, yields the most accurate and trustworthy measurements.

Table 1.1: Typical Configuration of Airbag Control Unit

Parameter	Setting	Comment
ACU Vendor	SiemensVDO	now ContiVDO
Main-μC	STMicroelectronics ST10	16-bit controller
Safety-μC	STMicroelectronics ST7	8-bit controller
Clocking Main-μC	32 MHz	–
Clocking Safety-μC	16 MHz	–
EEPROM	2 kByte	–
RAM	8 kByte	–
ROM	128 kByte	blockwise flashable
Energy Autarky	min. 150 ms	via capacitor charge
Internal Sensors	g_x, g_y	$\pm$ 50 g, Piezo-electric
Main Contact	75 Pins	Sumitomo dual-lock

1.2.2 Crash Sensors

Crash sensors are the central measurement components for detection of an impact. In general, crash sensors of an airbag control system can be categorized in:

- External crash sensors in the front part of the vehicle (cf. figures 1.2 and 1.3). These so-called *up-front* sensors are micromechanic devices which measure an acceleration based on Piezo-electric effects. In a front crash scenario, the up-front sensors are closest to the place of impact, thus they detect the collision first and send the digitalized measurement values to the ACU. Typical detection ranges are $\pm$200 g.
- External crash sensors on the sides of the vehicle. With the development of side- and window-airbags, additional measurement devices for detection of a side impact became necessary. As shown in figure 1.3, sensors are installed in the front doors and on the vehicle body close behind the rear doors. The devices in the doors are typically pressure sensors, which are equipped with a membrane to measure the increase in pressure inside the door in case of an impact, for example in ranges from 50 to 116 kPa. The side sensors on the vehicle body are using the same measurement principle as the up-front sensors, namely Piezo-electric elements for detection of accelerations of $\pm$200 g.

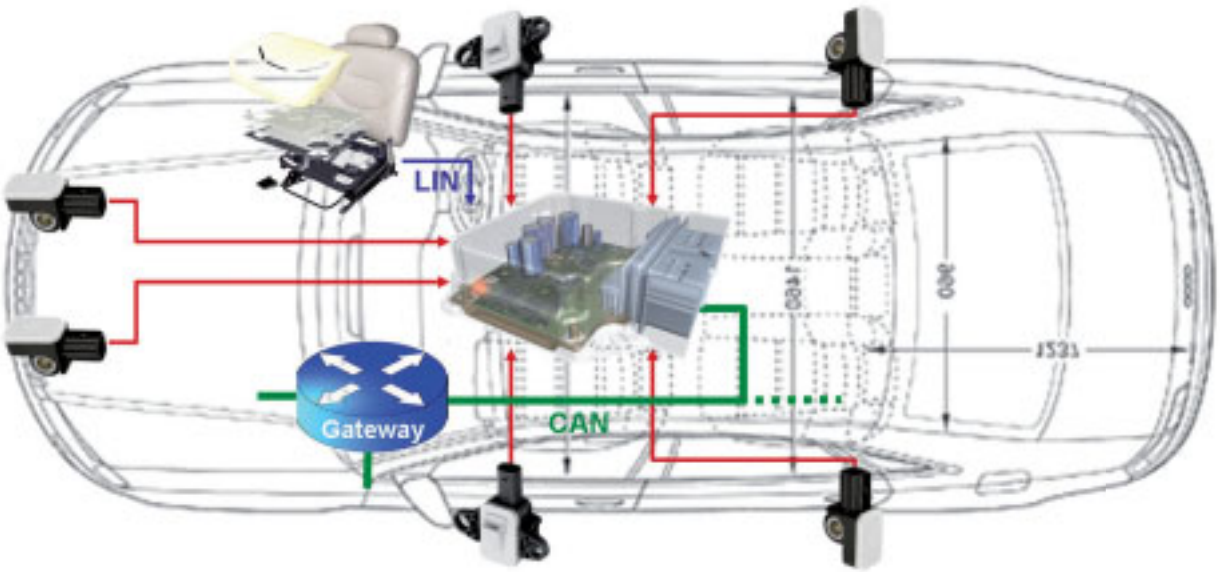


Figure 1.3: Placement and Interconnection of Passive Safety Electronics

- Internal crash sensors on the board of the ACU. For validation of signals from external up-front and side sensors, acceleration-based sensors are used. As the impulse of the impact is much lower in the middle of the car than at the damaged outer parts of the vehicle body, a significantly lower detection area of ± 50 g is sufficient for internal devices. For detection of dangerous side-slip or roll-over situations, yaw-rate sensors are employed, i.e. combined acceleration sensors for x-, y- and z-direction or liquid-filled sensors with a movable gas-bubble.

Each external sensor is connected to the ACU by a dedicated wire. The analogue measurement samples are digitalized by an application-specific integrated circuit (ASIC) in the sensor and, together with sensor status information, periodically transfered to the ACU. Widespread up-to-date protocols for sensor data transmission are the Peripheral Acceleration Sensor 4 (PAS-4) protocol from Bosch for acceleration sensors or, for pressure sensors, the PEGASUS protocol from SiemensVDO/ContiVDO. Here, Manchester-coded data words of 10 - 12 bits length are transfered over a twisted-pair current interface, typically every 250 μs, i.e. with 4 kHz clocking. The Peripheral Sensor Interface 5 (PSI 5) is supposed to be the upcoming standard for communication between crash sensors and ACU [46], the launch is planned for 2010.

1.2.3 Airbags and Belt Tensioners

The actual protection for occupants is achieved by appropriate activation of protection means in a crash, namely the airbags and the pyrotechnical belt tensioners as depicted in figure 1.4. An airbag is a textile cushion with a volume from about 15 liters (side-airbag) to more than 100 liters (front-airbag passenger side). To match the timing constraints of only a few ten milliseconds for opening the airbag completely, pyrotechnical processes are employed to generate the adequate amount of gas and to inflate the cushion. The same holds for the pyrotechnical belt tensioners, which have to reach force-levels of about 4 kN for tightening the passenger in the seat and thus to suppress a forward movement against the steering wheel or the dashboard. Upon detection of a collision with a certain severity, the ACU sends a *firing current* to the relevant protection devices. For example, in a front-crash with a speed of at least 26 km/h, the front-airbags and the belt tensioners for driver and front passenger are activated. The firing current triggers Natriumazid pallets in the airbag module to oxidate and to generate Nitrogen, which inflates the cushion. In the belt tensioner, Nitrocellulose is fired and the resulting portion of gas rewinds the belt mechanics via iron bowls.

Figure 1.4: Side-Airbags, Front-Airbags and Belt Tensioner [50]

1.2.4 Functional Safety of the Airbag Control System

Even though airbags and belt tensioners yield reasonable protection for occupants in a crash, an unmotivated deployment of these pyrotechnical devices in a non-crash driving situation remarks a serious threat for the passengers. Thus, considerable effort is spent to avoid misuse scenarios. First and foremost, the ACU relies on both external and internal sensor measurements for crash detection. In figure 1.5, this concept of *plausibility* is indicated by the green-colored external

and internal sensors serving as combined input to the ACU, where each sensor type can be used to validate the measurements of its counterpart. For instance, if a defective upfront sensor generates corrupted data reporting a front-crash, the ACU will detect its malfunction by validation of external sensor data with internal measurements.
The second important concept for functional safety of the airbag control system is the *redundancy* in the dual controller layout of the ACU (colored in blue in figure 1.5). Main- and safety-μC monitor the correct operation of each other by hardware and/or software watchdog mechanisms and evaluate sensor data redundantly. Only a timely- and event-correlated firing decision of both microcontrollers – indicated by the logical "AND" in figure 1.5 – will activate any firing stages. This concept prevents the system from erroneous processing of correct (external and/or internal) sensor data if one of the controllers comes into a malfunctional operation mode.
The third strategy which contributes to safe and reliable system operation is the *safing* concept. The main-μC evaluates sensor data from external and internal sensors permanently. Thus, the measured values show a certain plot over the time axis, resembling the currently measured acceleration or pressure. In case of a collision, sensor data values are expected to deviate from normal measurements significantly in terms of an instantaneous, steep increase of measurements due to the impact. Within the safing concepts, thresholds are implemented, which the sensor data must exceed with a specified increase or for a certain time interval before any pyrotechnical means are activated. This prevents from undesired deployments, caused by steep but rather short peeks in acceleration of the whole vehicle body, e.g. on rough roads, or by longer but rather low impulses, for example from low-speed contact with the bumper of another vehicle while parking.

1.3 Future Vehicle Safety Approaches

The passive safety systems for occupant protection are nowadays on a high level of effectiveness and functional safety. Thus, vehicle safety strategies based solely on passive airbag control systems and stiff vehicle bodies will hardly lead to significant improvements in this field. As a consequence, lots of effort is spent on novel developments for a further reduction of road traffic fatalities.

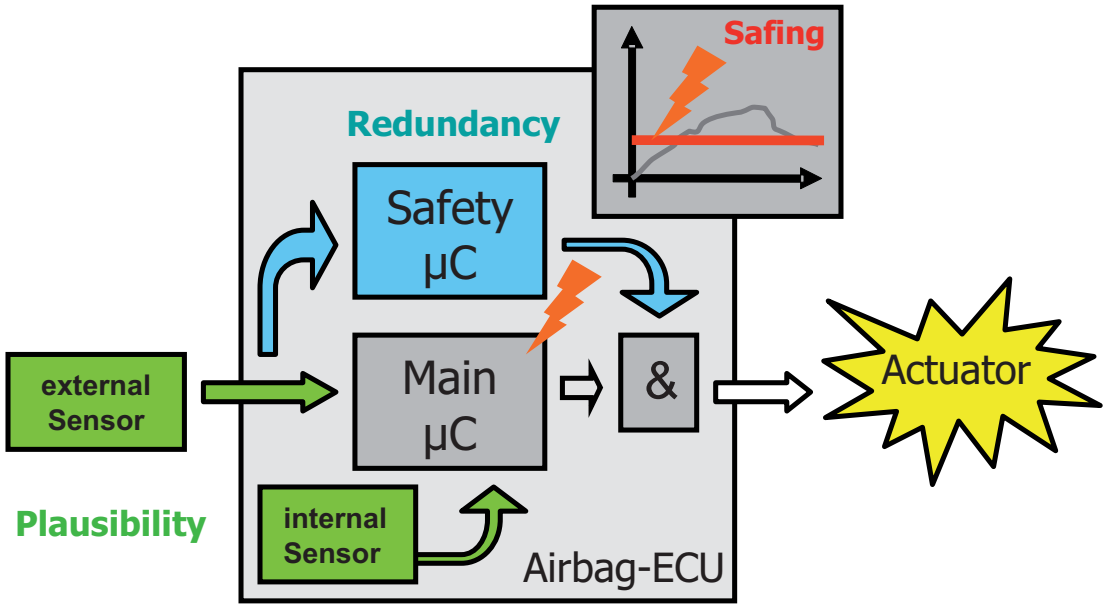


Figure 1.5: Functional Safety Concepts for Airbag Deployment

1.3.1 The Intelligent Car

A promising approach for increased safety for both occupants and other road users – especially vulnerable traffic participants like pedestrians or cyclists – is an enhanced capability of the individual cars in environment perception and communication with the surrounding traffic area. Several joint research projects from government, universities and industry, like *PreVent* [55] or *SAVE-U* [15], cope with such an "intelligent car". A possible road traffic scenario together with some intended applications and the corresponding time line of activation is depicted in figure 1.6. Individual cars perceive information about the road traffic situation and/or possible threats, obstacles or other road users via:

- *Car-to-X* communication. This approach is based on wireless data transmission, where the "X" is a placeholder for either communication with other road traffic participants – *Car-to-Car* – or with fix installed devices, called *Car-to-Infrastructure* data exchange, e.g. with communication units from municipal traffic management systems. The wireless data transmission might be realized either using ad-hoc *wireless local area networks (W-LAN)* as proposed in the SimTD project [56], or based on mobile communication standards, e.g. *Universal Mobile Telecommunications System (UMTS)*, as employed in the CoCar research project [7].

- Environment Sensor Systems. Well-known from military and avionics, technologies like radar, laser, vision and ultrasonic enable the car to perceive information on the environment [41], [40], [34]. For purposes of active safety and drivers comfort, some of these devices are already available, e.g. the radar-based *Adaptive Cruise Control (ACC)* and *Lane Change Assist (LCA)* or the ultrasonic-based *Park Distance Control (PDC)*. In future vehicle safety approaches, information from environment sensor systems is used to provide passive safety systems with a-priori knowledge about a risky situation, which is seen as a promising instrument to increase the efficiency of passive protection means.

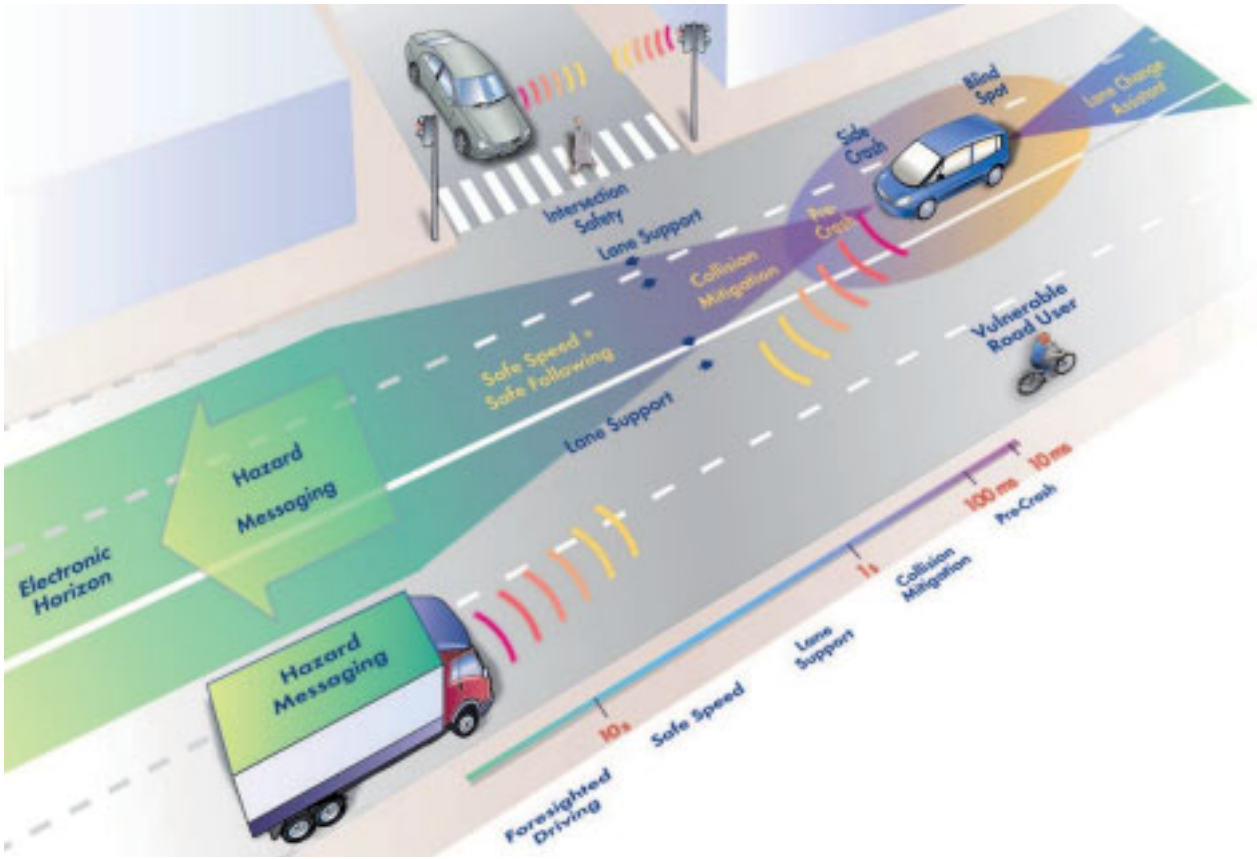


Figure 1.6: Future Vehicle Safety and Road Traffic Scenario [55]

1.3.2 Precrash Systems

If environment information is evaluated and the on-board electronics react on a dangerous driving situation in order to avoid fatal consequences like a collision,

a so-called *precrash system* is implemented. Several strategies for engagement are applicable, for example, if roadside units communicate an icy or slippery surface to the Car-to-X receiver, the driver can be warned optical or acoustical. A more autonomous system reaction would be an engagement in the braking system, for instance, if the radar sensor of the ACC detects an obstacle, e.g. a vehicle remarking the end of a traffic jam on a highway[1]. If the driver does not activate the brakes and remains heading for collision, the ACC will autonomously trigger a full brake application to avoid a crash.
Basically, the visible or sensible actions of a precrash system are coordinated by components of active vehicle safety and are intended to avoid the collision by any means necessary. However, driving situations may occur in which, in spite of all warnings or engagements from active safety devices, the collision is unavoidable. This is the point where passive safety components come into the plot of precrash systems. Active safety components, like the ACC or the *Electronic Stability Program (ESP)*, can communicate information on the hazardous driving situation to passive safety components, especially to the airbag control unit. This information is valuable knowledge for the ACU. Without sensor data from active devices, the airbag control is able to react on a collision only when external sensors detected the acceleration of an impact, sensor signals are validated, algorithmic thresholds are exceeded by the measured values and both microcontrollers decided in favor of activation of protection means. Encompassing precrash information allows for various strategies to increase the efficiency of passive safety concepts:

- Lowering of algorithmic thresholds for deployment. If a crash is to be expected within a certain time interval, the robustness of the deployment algorithm in the ACU might be traded off for a faster deployment decision. If the crash happens, lowered thresholds will be exceeded sooner by crash sensor acceleration signals, saving a few milliseconds in signal integration and firing stage activation.

- Activation of irreversible protection means. Based on trustworthy information on an upcoming, unavoidable collision, pyrotechnical components might be activated shortly before the impact. This allows to trigger conventional systems more effectively, e.g. by installing airbags with larger volume, which can be inflated completely in the precrash phase, or to em-

[1] This is one of the scenarios on which considerable effort for effective precrash systems is spent. Just think of the horrible crashes when trucks collide with high speed into a highway traffic jam.

ploy new pyrotechnical devices like airbags on the outside of the vehicle body, providing additional deformation space if inflated before a collision. Of course, such concepts require a high degree of reliability in environment information to avoid misuse.

- Activation of reversible components. For example, motor-driven belt tensioners allow to tighten the front passengers in the best, upright seating position, reducing the forward movement from the very beginning of a collision [33]. As the actuators are designed for several activations during the lifetime of a car and the effects are less immanent than in case of pyrotechnical components, the misuse criticality is not that high. Of course, permanently unmotivated activation of these components annoys customers and thus should be avoided by respective functional design of the precrash system.

Research on and development of such concepts is a big issue in vehicle safety for both car manufacturers and suppliers. Figure 1.7 depicts some exemplary applications and the involved, networked active and passive safety components. The examples are selected from the current portfolio of so-called *combined active and passive safety (CAPS)* systems from the German technology venture Robert Bosch GmbH. More information on the concepts can be found under the cited weblink.

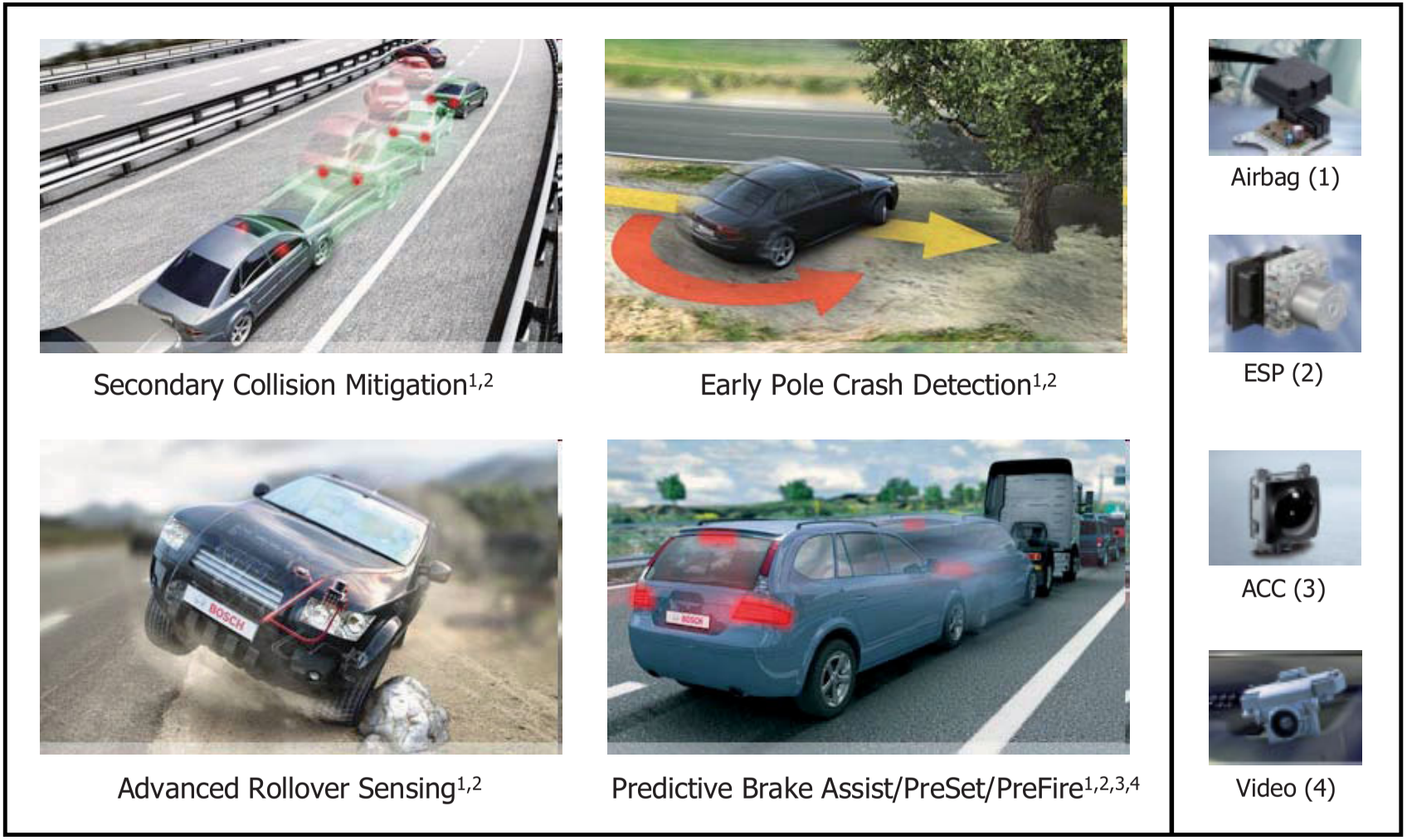


Figure 1.7: Examples of Future Vehicle Precrash Functions and Employed Safety Devices [50]

1.3.3 Time-Critical Precrash Data Transfer

The timeline in figure 1.6 depicts a possible plot of activation of various intelligent applications. Obviously, functions focusing on collision mitigation and precrash concepts are triggered very late in a dangerous driving situation, i.e. typically in a range from 1 second down to 10 milliseconds prior to a critical event. For an interplay of active and passive safety devices, this implies a forced flow of data, coping with such strong timing demands. Imagine a radar sensor detecting a threat in the drive lane, for which the vehicle is on target for with a constant speed. If the information from the radar can be transfered to any passive safety device just in time, precrash applications and appropriate protection means can be employed efficiently before a collision.
If it takes too long to pass the environment information from the sensor to the actuators, the crash might occur even before any passive device could be informed about the threatening situation and any protective actions could be taken. At that point, the data transfer between active and passive safety systems comes particularly into play. The effectiveness of networked precrash applications is strongly related to the speed and the performance of information exchange inside the car. The short precrash timeframes before a collision require a predictable real-time behavior of data transmission, incorporating all relevant networking constellations and operation scenarios.
Measures like end-to-end transmission delay or jitter allow for a determination, whether a given communication topology is qualified for transfer of safety-relevant data in highly time-critical operation scenarios.

2 Related Work

2.1 Measurements

For real-time data transmission in a small-sized CAN network, *Navet and Song* employed both a model-based (analytics and simulation) and a prototype-based evaluation in [43]. Analytics are applied to derive worst-case response times, simulation models are built based on both queuing networks and finite state machines. The results from analytics and simulation are compared with outcomes from monitoring of a distinct prototype measurement setup.
The measurement study from *Ferreira et al.* focused on determination of reasonable bit error rates for CAN [16]. By bit stream comparisons in a dedicated topology with a 30 m CAN cable, the error rates for benign, normal and aggressive environments could be determined.
Pallierer et al. presented in [45] a generic toolbox for measurements in FlexRay and CAN communication systems. The focus is on systematic testing, data recording and evaluation, e.g. regarding system startup, operation and fault tolerance.
The approach of *Albert and Gerth* [1] concentrates on a comparison of real-time capabilities of CAN and the time-triggered derivative TT-CAN. The presented test setup works off-board and considers only two communicating *Electronic Control Units (ECUs)*. For this configuration, the response time in an asynchronous on-event communication scenario is measured and conclusions about the compliance with real-time demands are drawn.

2.2 Simulation

An approach based on the *Unified Modeling Language (UML)* for modeling of automotive applications is presented by *Schröder and colleagues* in [54]. The focus of this proposal is on requirement engineering methods and statechart annotations

for software tasks in ECU networks, no simulation model is built from the UML constructs.
FlexRay star topologies are in the spotlight of the simulative investigations of *Ortega et al.* in [44]. The electro-magnetic compatibility is investigated for FlexRay with 10 Mbps data rate. Analytical outcomes from Impulse-Reflectometry and results from a circuit simulation are validated vice versa. However, the emphasis of the article is on interference on the physical layer, not on higher layers of end-to-end data transmission. A similar approach is presented from *Kraft* in [31] for physical layer simulations in a CAN network with several ECUs.
Castelpietra et al. developed a modular modeling and simulation technique [6]. Both hardware and software architecture are modeled in a proprietary modeling environment, called *Carosse-Perf*. This naturally decreases the level of exchangeability, maintainability and compatibility compared to a modeling approach based on a commonly used, intuitive technique like UML, as proposed in this thesis. In a case study, the simulation results for end-to-end response times in a given ECU network are compared to outcomes from analytical worst-case performance evaluation.
Modeling and simulation of a distributed embedded real-time system using the language *SystemC* is performed by *Samii, Rafiliu, Eles and Peng* in [52]. The worst-case response time, as obtained from simulation of a system composed of CAN and FlexRay buses, is compared to results from worst-case analytics in order to derive the "degree of pessimism" of analytical upper bounds, i.e. how much the computed thresholds exceed the simulated maxima. The number of scenarios, which have to be simulated in order to achieve a satisfying coverage, is minimized by application of various algorithms for exploration of a given design space.

2.3 Analytical Modeling

In [57] and [63], *Thiele et al.* applied analytical modeling approaches to systems with hard real-time demands. The scope of these works is on interface-based system design and preemptive scheduling of ECU and processor tasks with fixed priorities, according to commonly employed task scheduling strategies. The proposed *Real-Time Calculus* incorporates modeling elements from Network Calculus, e.g. the service curve of the system is built priority-dependent as done for CAN in this thesis.

The modeling technique of *Response Time Analysis (RTA)* is used by *Tindell and Burns* in [58] for analytical investigations on a CAN bus. It is determined, whether for a given schedule of tasks on ECUs in the system, deadlines and real-time demands can be fulfilled. The scope of this proposal is extended towards an improved priority scheduling policy and an automated assignment of task and message periods by *Burns* in [12] and by *Davare et al.* in [11].
Krakora et al. developed a system representation of CAN as *Timed Automata*, which can be found in [32]. CSMA/BA media access, the CAN bus and an ECU with CAN transceiver and application processes are resembled in notions of automata. Compliance to pre-defined real-time demands is derived by computing worst-case response times according to [58] and by checking the model properties using the model checker UPPAAL [59].
Ernst et al. present in [19] a software-based exploration and optimization approach, which can be employed in the design phase of a CAN bus. By design space exploration and optimization, the tool *SymTA/S* [24] offers formal methods for studies on the feasibility of given task schedules and capabilities of resources in a CAN bus. A case study for automotive real-time applications is presented in [13]. In [47], the formal methodology is enhanced in terms of analysis of the *design sensitivity* of an embedded real-time system towards variation in parameters. Several input is considered and alternated, like task periods, jitter and communication load. The sensitivity of the system is computed as the so-called *feasibility slack*, obtained from the variation of optimization properties like minimum execution time or maximum communication volume.
In order to improve the real-time behavior of CAN data transmission, *Richardson et al.* introduce a methodology to detect transient surges due to retransmissions or sporadic, a-periodic transfer [49]. However, to implement a dynamic priority scheduling of ECU tasks according to the *Earliest Deadline First* algorithm, the CAN frame identifier would have to be partitioned into a 4-bit dynamic priority part and a 7-bit static priority part, reducing the number of a-priori assignable IDs from 2048 to 128.

The drawback of the analytical proposals mentioned so far is, that a static a-priori schedule for *all* tasks and messages in the system is required for evaluation of real-time properties. For two reasons, such a holistic schedule is hardly achievable: First and foremost, most of the tasks of an operating system in an ECU are not accessible from outside – including the schedule, the activation period or the processing strategy. This area of the controller is typically intellectual property of

the third-party supplier manufacturing the ECU. Most of the internal software is rather in a black-box, except for the data which is explicitly communicated to a bus, a testing device, etc.
For the second, even if the internal task scheduling in every ECU is known, the controllers along a CAN bus wake up asynchronous to each other, requiring individual, varying timespans. This assumption will be emphasized by the results from the measurements in this thesis, presented in chapter 4. Hence, a valid *overall* schedule of all tasks on all ECUs can never be anticipated and at least the input parameters regarding local and global task activation times and scheduling yield a certain fuzziness in the results.
To overcome this, the Network Calculus evaluation procedure proposed in this thesis requires only the statically assigned CAN identifiers and the specific cycle times at which each message is sent as input data. The global, bus-wide schedule of traffic does not have to be known to obtain reasonable upper delay bounds for each priority class in the system.

A completely different approach is to model the system as a queuing network, as recommended by *Roberts and colleagues* in [51]. For ATM traffic flows from several sources – but without priority scheduling – a $\sum D_i/D/1$ queue is proposed for adequate modeling. However, the solutions, which are determined using stochastic methods, are valid for the totality of all jobs, not for individual classes from individual sources. Furthermore, upper bounds are provided for the probability that the backlog in the system exceeds a certain threshold.
The application of Network Calculus in this thesis does not consider any stochastics. Deterministic upper bounds are obtained for each individual priority class, not only for the totality of jobs, messages, customers, etc. Moreover, the CAN bus also differs from the $\sum D_1/D/1$ queue in its strict priority scheduling.

3 In-Car Communication System

Today, an upper class vehicle is equipped with up to 100 ECUs, serving for various purposes of entertainment, infotainment, drivers' assistance and comfort, or applications of active and passive occupant protection. Besides the increasing power consumption – just think of the current discussion on CO_2 emissions – the communication and data exchange between the single control units is one of the most critical tasks to be considered when expanding the electronic functionality inside the car. What all applications, regardless of the actual criticality, have in common, is the demand for robust, reliable and efficient data transmission between incorporated ECUs. To interconnect these dozens of embedded controllers, several bus systems have been established in automotive environments. The most important representatives are *Controller Area Network (CAN)* [25], *FlexRay* [18], *Media Oriented Systems Transport (MOST)* [42] and *Local Interconnect Network (LIN)* [60]. They differ in data rate, media access and multiplexing schemes and other, mostly hardware-related peculiarities. Typical structures of up-to-date automotive in-car communication systems with a central gateway or multiple, cascaded gateways and various buses are depicted in figure 3.1.

3.1 CAN - Controller Area Network

CAN development has been started in the 1980's by Robert Bosch GmbH, aiming to design a bus system for the specific needs of automotive applications. The resulting CAN bus is now standardized as ISO 11898 [25]. It uses a differential serial line architecture, cf. figure 3.2, with *dominant* and *recessive* bits, where a dominant bit represents a logical 0 and a recessive bit a logical 1. An idle bus has recessive level. Due to the open collector logic, if one station on the bus sends a dominant bit while another controller sends a recessive bit, the dominant bit wins, i.e. the bus is considered as logical 0. This feature allows for use of a bitwise

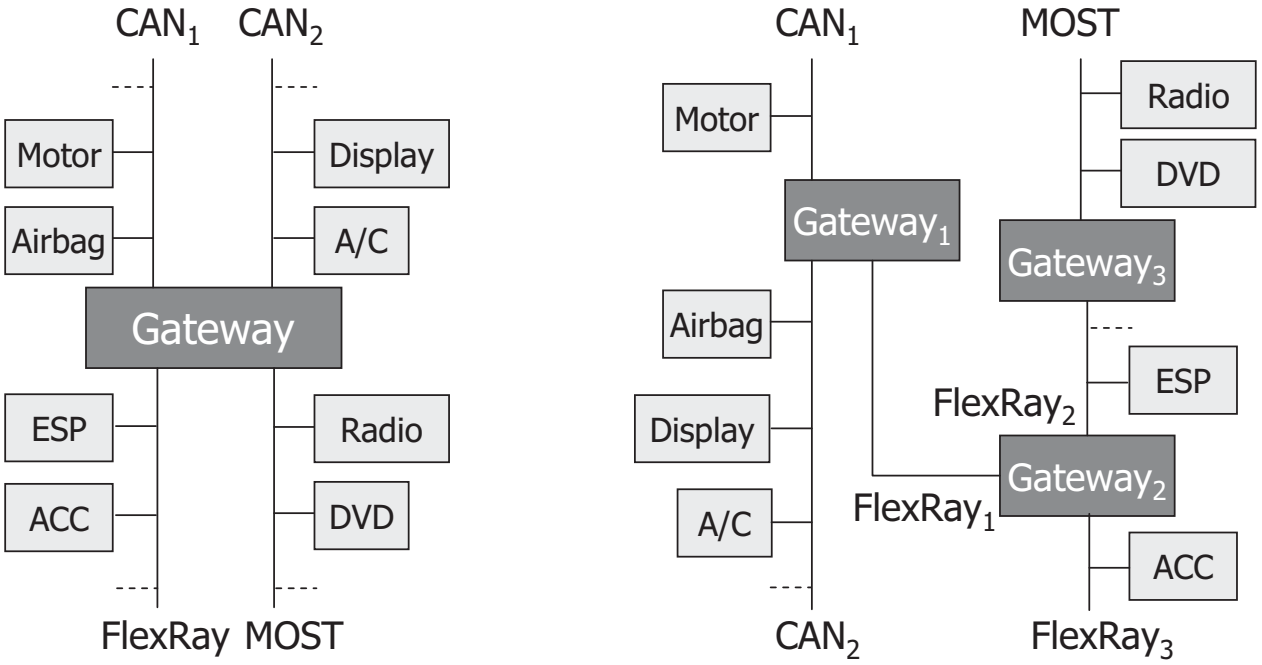


Figure 3.1: Network Topologies: Central Gateway (left) and Cascaded (right)

arbitration scheme for medium access, often called *Carrier Sense Multiple Access with Bitwise Arbitration (CSMA/BA)*. Using this mechanism, each station listens to the bus while sending data. If a collision occurs, where one station tries to send a recessive bit but receives a dominant bit, it will notice that another station is sending simultaneously and will stop its own transmission immediately. This makes the arbitration non-destructive, since the station sending the dominant bit can continue to send without any negative effects on the bus, while the station sending the recessive bit remains silent from the time on where the collision has occurred. A retransmission of the interrupted frame is triggered automatically.

Before sending, each controller listens to the bus and starts sending only if the bus has been recessive for at least 6 bit times (carrier-sense phase). A *Non-Return-to-Zero (NRZ)* encoding is used for the line encoding of the bits, where the sender inserts a complementary stuff bit when no change in the logic level has occurred over 5 successive bits. These stuff bits are removed automatically by the receiver of the message. This mechanism provides a base for synchronization and assures that a potential sender receives at least one dominant bit during the 6 bit times of carrier-sense phase if another station is sending.
CAN does not employ explicit sender or receiver addresses, but uses unique message identifiers to describe the content of a CAN frame. The frames are

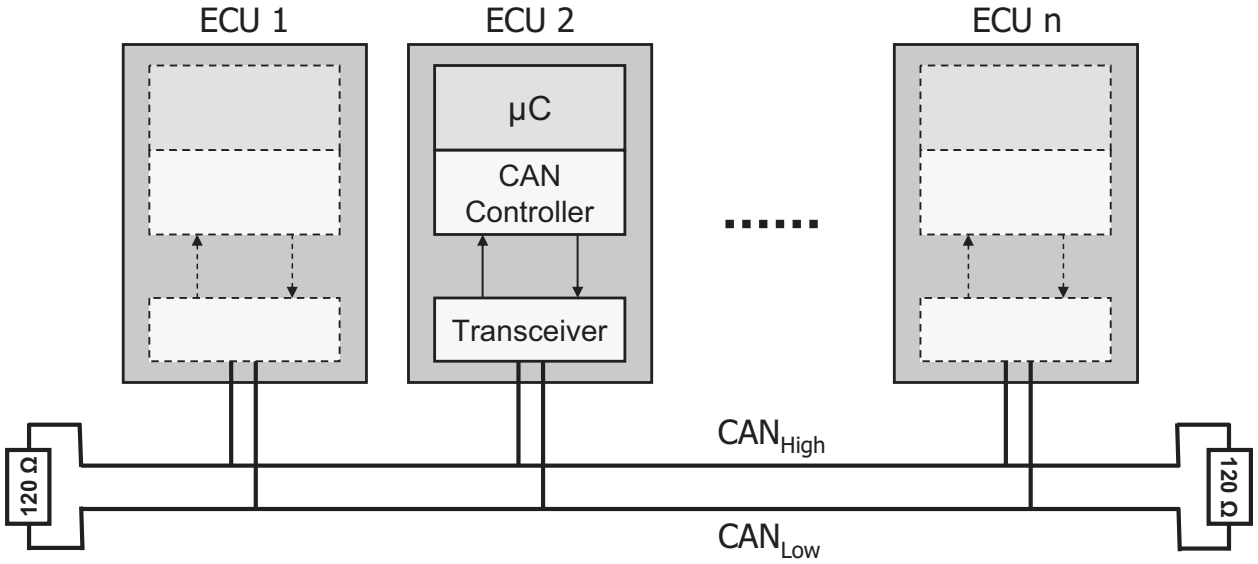


Figure 3.2: CAN Serial Line Architecture

broadcasted on the bus and each station can decide if the message content is relevant by examining the message identifier of a received frame. The identifiers have to be assigned statically during the design phase of the bus system to avoid ambiguity in the interpretation of the frame content. A global view on the complete communication system is needed in this process.
There are two variants of CAN frames: standard frames with 11-bit message identifiers and extended frames with 29-bit identifiers. Both can coexist on the same bus. The payload can be of variable size, but throughout this thesis, all frames are standard CAN frames with 11-bit message identifiers and are assumed to contain always 64 bits of payload. Thus, when considering a maximum number of 19 stuff bits and including 3 bit times of inter-frame space (IFS), the maximum frame size is 130 bits.
The transmission of a CAN frame starts with one dominant start bit, immediately followed by the message identifier from the most significant bit to the least significant bit. The structure of a standard CAN frame is shown in figure 3.3.

Due to the media access scheme as described above, the message identifiers create an implicit hierarchy of priorities. If more senders start to send simultaneously, the sender transmitting the frame with the highest message identifier has to send a recessive 1 bit that is overwritten by a dominant 0 bit first, due to the binary encoding of the message identifiers in the frame header. While listening to the bus during the send process, it will notice the collision – dominant bus while sending

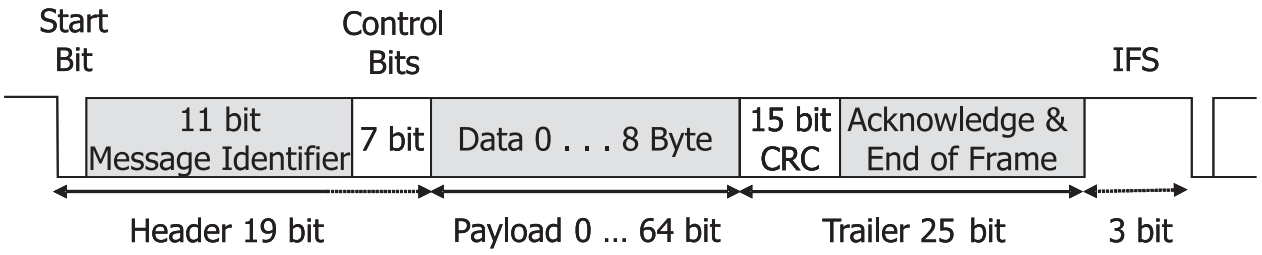


Figure 3.3: CAN Frame Structure

a recessive bit – and stop sending. This process continues until only one sender remains sending. This is always the one with the lowest message identifier and thus with the highest priority [37], [14].

3.2 FlexRay

FlexRay [18] counters the increasing demands for transmission capacity, network complexity and reliability. It is a field bus system, implementing a deterministic *Time Division Multiple Access (TDMA)* media access scheme, which guarantees to provide each connected ECU with transmission capacity in its dedicated time slot. Two physical channels are available, each one offering a maximum data rate of 10 Mbps. Transmission can be performed either fully redundant on two channels in parallel, or both channels can be used for data transfer individually, yielding an increased overall data rate of 20 Mbps. A communication cycle in FlexRay is organized as depicted in figure 3.4.

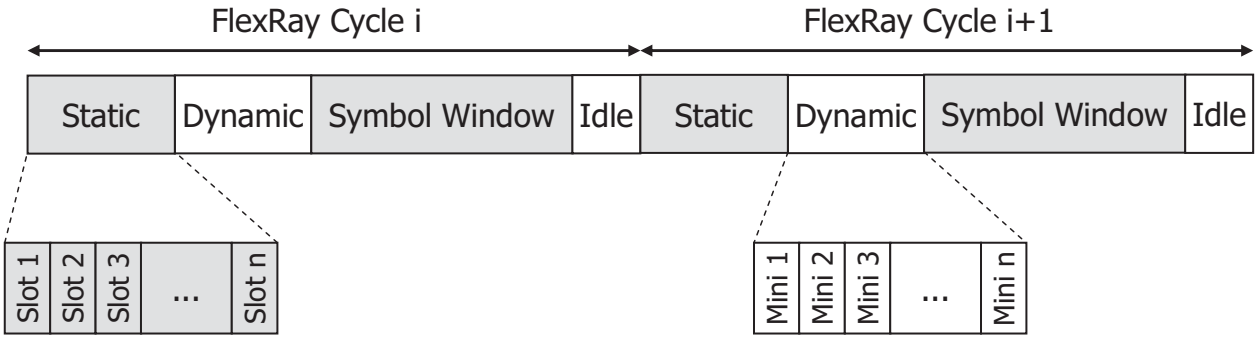


Figure 3.4: FlexRay Communication Cycle

In the static segment, each ECU is allowed to sent data in its dedicated time

slot. The following dynamic segment implements *Flexible Time Division Multiple Access (FTDMA)*, where, based on a-priori assigned priorities, ECUs are able to send additional data in mini slots on demand. In its static FlexRay time slot, an ECU can send several signals, which are packaged as payload in a FlexRay frame. Figure 3.5 shows a FlexRay frame with header and trailer fields and a maximum of 254 byte payload.

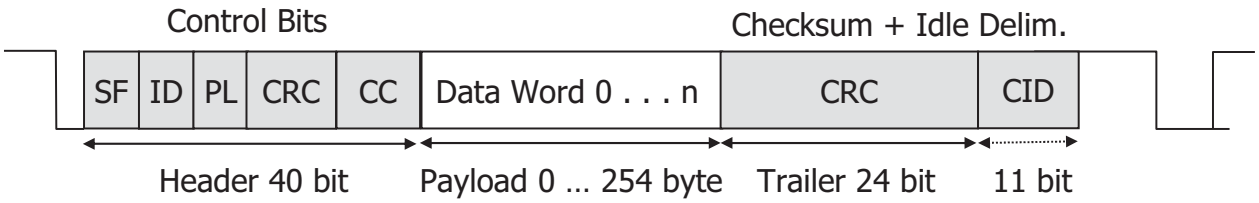


Figure 3.5: FlexRay Frame Structure

As in any TDMA-based communication system, a common time base for all sending and receiving stations is indispensable. FlexRay provides a decentralized synchronization of all ECU communication controllers, where a single controller derives the global network time from a virtual global clock by evaluating timing information from several other ECUs.

3.3 Automotive Gateway Architectures

An automotive gateway is an important node in the in-car communication system. It connects buses, routes messages and performs network management tasks, e.g. monitoring states of operation of ECUs or whole bus segments. With a dedicated controller and transceiver for each bus system the gateway is connected to, it performs just like any other ECU on the respective bus, e.g. regarding CAN or FlexRay media access. Figure 3.6 depicts a possible structure of an automotive gateway.

According to an internal routing table, the gateway receives and forwards messages from and to buses, where incoming messages can be processed in various ways. As commonly applied for routing in the Internet or in local area networks, automotive gateway routing mechanisms can basically be categorized into routing by periodical scanning of ports of connected bus systems – so-called *cyclic polling* – and into strategies which perform an event-based routing, e.g. triggered by an

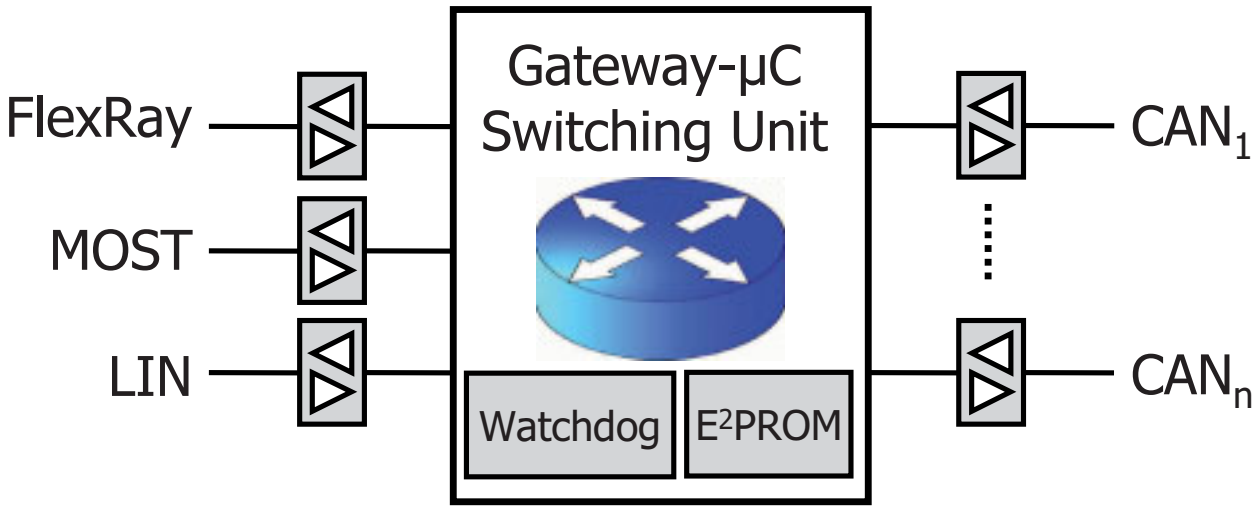


Figure 3.6: Typical Automotive Gateway Layout

interrupt if at any of the ports a message arrives. Figure 3.7 depicts an exemplary cyclic polling mechanism in a gateway with three connected buses. The incoming ports "Bus i in", $i \in \{0,1,2\}$ are periodically checked for newly arrived data, comparable to a time-dependent token which is passed from one port to the next. If the port contains data, the ordering strategy in the port determines which message is to be taken out actually. For example, for *first-in-first-out (FIFO)*, the oldest message is routed whereas for *priority queuing*, the message with the currently highest priority is forwarded. The timeout for proceeding from one bus port to the next and the number of messages to process from the port within one cycle can either be fix or adjusted individually, e.g. according to the communication load on the source bus or the maximum number of messages to be stored. Anyway, both parameters should be chosen such that an overflow and possible loss of messages in the respective port is avoided.
In figure 3.8, the interrupt-based routing mechanism is illustrated. Again, three buses are connected to the gateway via the incoming ports "Bus i in", $i \in \{0,1,2\}$. Instead of cyclic polling and processing of data, arriving messages trigger routing and forwarding directly, indicated by the single queue in which all data is stored and processed from. Again, FIFO ordering in the central buffer is applicable as well as other, more sophisticated ordering strategies, like priority-based routing.

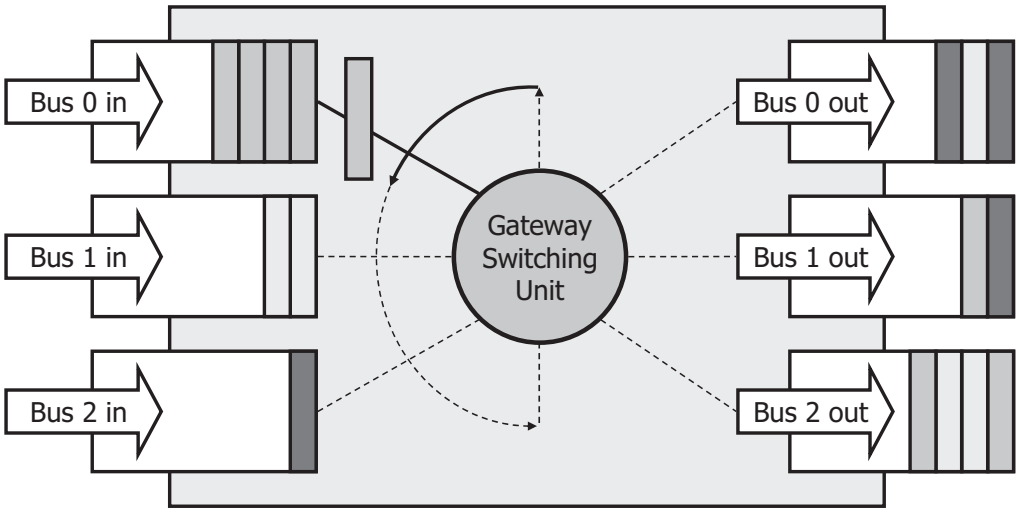


Figure 3.7: Cyclic Polling Routing Functionality

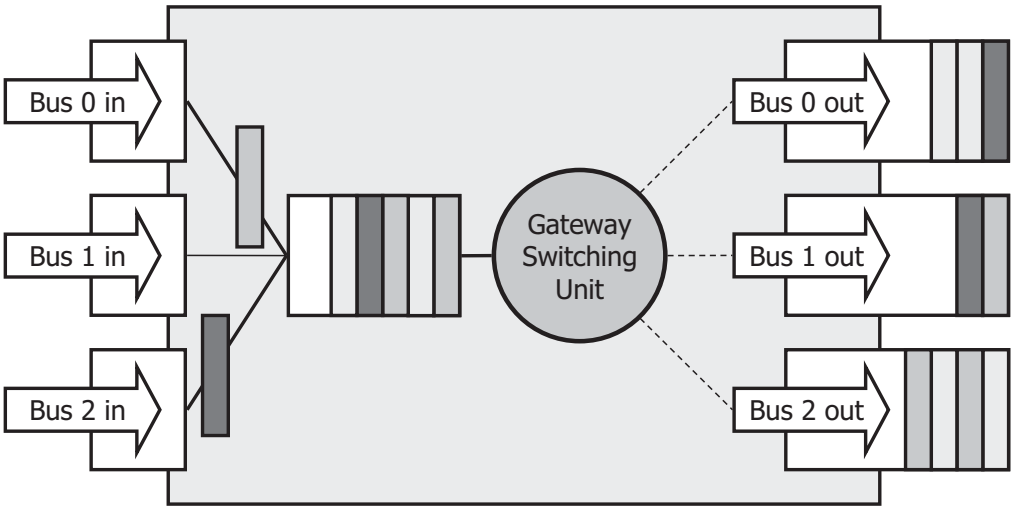


Figure 3.8: Interrupt-based Routing Functionality

3.4 Other Communication Technologies

There exist several other automotive bus systems, like LIN, MOST or Ethernet-based field buses. Due to system performance and peculiarities in communication protocols, these technologies are rather employed for comfort, entertainment or master-slave data exchange than for time- and safety-critical applications. Thus, the following investigations are restricted to network topologies consisting of CAN, FlexRay and a central gateway.

4 Prototype Measurements of In-Car Data Transmission

The following chapter presents results from prototype measurements which were conducted together with Bernhard Kloiber during his diploma thesis [30], supervised from July 2008 to January 2009 at the *Department of Safety Electronics of the Audi AG, Ingolstadt, Germany*. Both the measurement hardware setup and the software-based data evaluation strategy are introduced and applied to real-life issues of ECU operation and in-car communication. The methodical approaches and several outcomes of the investigations were published in [22] and [23].

4.1 Motivation

A modern car is a complex, electronic system-of-systems, composed of tens of ECUs. Communication and data exchange between these controllers is essential for manifold application areas of nowadays in-car electronics. While some investigations can easily be performed knowing the technical specifications of the controllers or the communication infrastructure, others require a detailed inspection of the system in real-life operation. For example, the CSMA/BA mechanism itself is implemented in every standardized CAN controller device, which thus can be assumed to provide the intended functionality of priority-based media access with bitwise arbitration. On contrary, technical specifications of many ECUs do not provide sufficient information on the startup behavior of the ECU, or on the frequency drift of the controller quartz, to reliably anticipate the actual performance in daily operation.
Determining such parameters is a necessary task for several reasons. On the one hand, a validation of the product quality and compliance to technical specifications is achieved by an evaluation of the measured behavior of the component. On the

other hand, important system parameters can be derived for classical performance evaluation approaches, e.g. realistic controller startup durations or frequency drift values may serve as input for a dynamic simulation model of the in-car communication system. Last but not least, real-life measurements are particularly suited to emphasize the assumption that theoretically expectable effects are really physically present in a system and are possibly of significant impact on the overall communication performance, e.g. priority-dependent delays at CSMA/BA CAN media access.
To achieve all this, a sophisticated measurement infrastructure is indispensable, composed of measurement hardware and appropriate software constructs for a reasonable and detailed evaluation of recorded data samples. Hardware devices and software constructs must be easy to adapt to actual communication settings, deployed ECUs and bus systems, and capable of yielding accurate and trustworthy results for a broad variety of impressive performance measures.

4.2 Measurement Hardware Setup

The object of study for all following investigations was an Audi A6 Limousine 3.0 TDI with a prototype communication system topology consisting of 1 FlexRay (10 Mbps), 5 CAN buses (500 kbps) and a central gateway. 28 ECUs are installed in total inside the car, like engine and airbag control, entertainment devices or driver assistance ECUs like ACC. To record in-car communication data, a CONDALO CCO DLIII data logger was used as described in [8] and depicted in figure 4.1.
The device comprises interfaces for 8 CAN buses, 1 FlexRay, 1 MOST, 1 LIN, 8 analogue and 8 digital line-in ports. It is equipped with an 80 gigabyte hard disk and an USB port for direct connection to a computer. The data logger was installed in the trunk of the car and connected to the central gateway by cable. Figure 4.2 shows the prototype measurement infrastructure. Each bus segment is recorded via its dedicated interface at the data logger. Incoming communication data is marked with a timestamp, which is generated according to the time of the internal processor clock of the device with an accuracy of 1 μs.

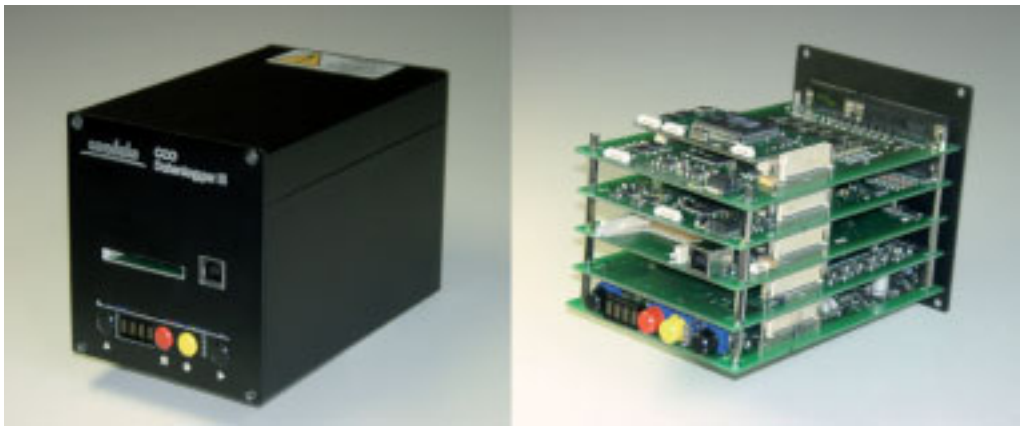


Figure 4.1: CONDALO CCO DLIII Data Logging Device [8]

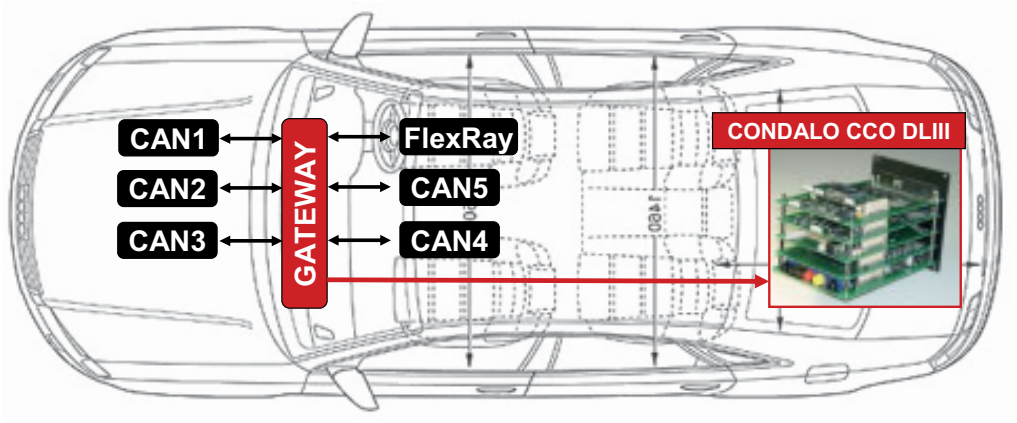


Figure 4.2: Prototype Measurement Infrastructure

4.3 Software Tooling for Data Evaluation

4.3.1 Communication Access Programing Language and VECTOR CANoe

The data recorded in various test scenarios was analyzed using the *CANoe* software suite from VECTOR [61]. One major aspect of data evaluation in automotive environments is to automate the evaluation process as far as possible, using appropriate source code for inspection of content in data frames, tracking messages along several branches in the network or determining actual communication startup times of ECUs. To handle gigabytes of logged data in an efficient way, evaluation routines were created using the C-based *Communication Access Programming*

Language (CAPL) [62]. The data processing strategy using CAPL programming constructs is illustrated by the flowchart in figure 4.3.

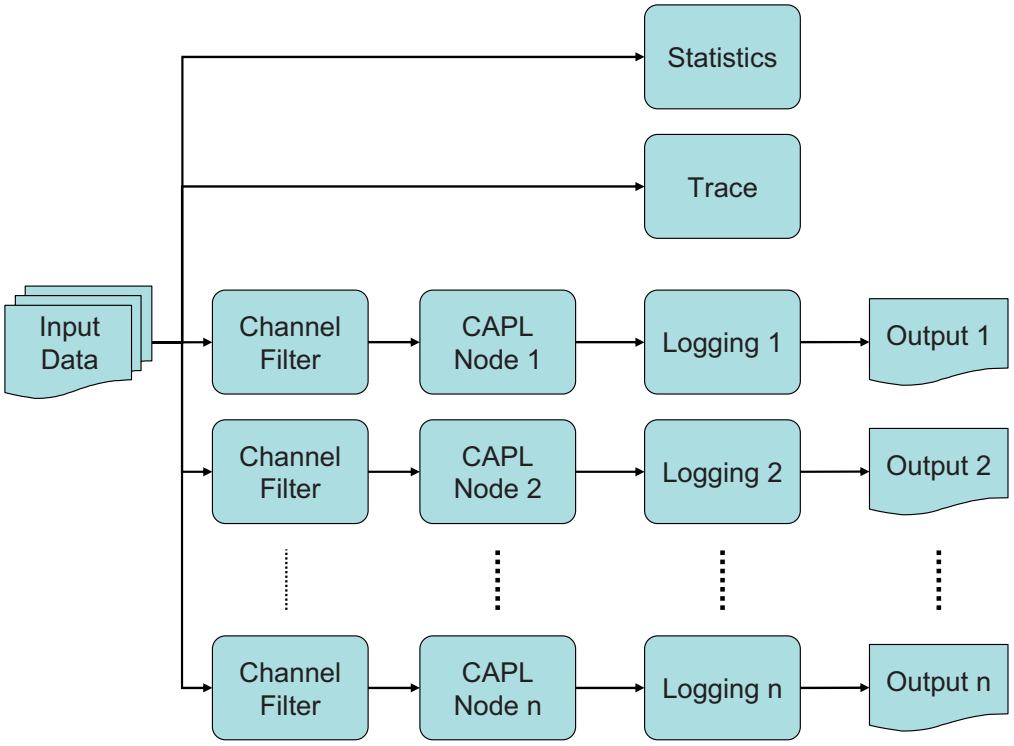


Figure 4.3: Processing of Measured Communication Data Samples in CAPL

The recorded data samples serve as *Input Data*. Processing of data is done along various branches. The block *Statistics* yields statistical performance measures of communication, like utilization of single buses or the overall amount of transmitted data. *Trace* allows for individual observation of any predefined subset of data transfer. Applying a *Channel Filter* restricts the measurement data to samples from certain bus segments. A *CAPL Node* contains the appropriate programming constructs with regard to the actual purpose of the measurement data evaluation, e.g. extracting timestamp information or inspecting information content of a CAN or FlexRay message. The *Logging* block is capable of collecting evaluation results while the input data file is processed. Finally, *Output* generates a summary of data evaluation and writes the results to an external file, e.g. in ASCII format.

4.3.2 ExpertFit®

A more detailed data analysis yields the probability distribution according to which values of measured data samples occur. This is valuable information, e.g. if the dynamic communication is to be simulated and appropriate input data modeling for stochastic values is required, or if other analytical techniques with statistic paradigms shall be applied to the measurement data. The statistical distribution fitting tool ExpertFit® [35] was employed to evaluate the type of the distribution as well as relevant distribution-specific parameters.
ExpertFit® is provided with measured data samples – as a list of comma-separated values, e.g. for delays or startup durations – and applies statistical methods to derive probability distribution functions that fit to the range and nature of occurring values in the set of measured data. The outcome is typically a ranking of suggested distribution functions with respect to the individual *goodness-of-fit*. The goodness-of-fit describes qualitatively how close a suggested distribution fits to the measured data samples, e.g. as ranking in terms of *good*, *borderline* or *bad*, together with relevant parameters and moments of the distribution like mean, variance, standard deviation or skewness.

4.4 Measurement Studies of In-Car Communication

4.4.1 Frequency Drift of Controller Quartzes

All measurements of ECU operation and in-car data transmission focus on aspects of *timing* in the distributed system of communicating controllers, namely startup durations, cyclic sending of CAN or FlexRay messages or end-to-end routing and communication delays. Hence, time bases are indispensable, either a global one like for FlexRay, or an individual time base in each CAN ECU or in the CONDALO data logger. In general, timing in an electronic device is based on oscillating crystals, where the oscillation of a quartz is used to derive a pulsing for microcontroller operation. Consequently, different quartzes can vary significantly in frequency, phase or temperature dependency of oscillation, leading to a so-called *drift* in timing of the respective ECUs compared to a virtual global clock. The drift is commonly denoted in *parts per million (ppm)* and can be either positive, which means the quartz is oscillating *faster* than a reference time base, or negative,

for a quartz oscillating *slower* than a reference time. For CAN and FlexRay data transmission, drift of ECUs might noticeably effect on communication as illustrated in figures 4.4 and 4.5.

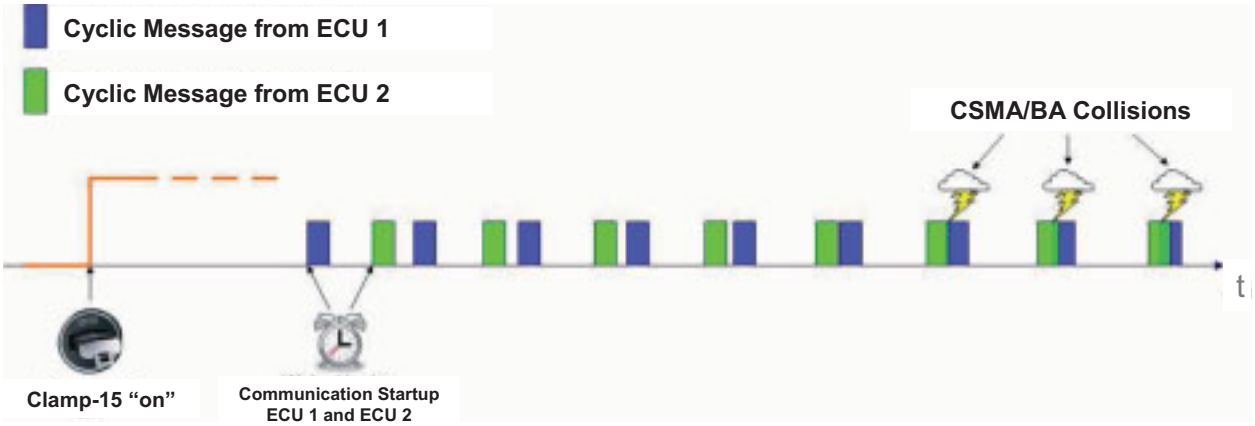


Figure 4.4: Effect of Frequency Drift of CAN ECUs [30]

As depicted in figure 4.4, two CAN ECUs (ECU 1 and ECU 2) start cyclic CAN communication at different points in time, such that initially the ECUs do not collide one with each other at CAN media access. Due to frequency drift of the controller quartzes, the ECUs will not keep this initial gap between cyclic communication, but rather move the time instances of data transmission towards each other. Hence, depending on the actual magnitude of frequency drift, collisions between ECU 1 and ECU 2 will occur sooner or later in the long-run of operation. Any collision at CSMA/BA media access contributes to transmission delays. Since a CAN bus system typically consists of up to several tens of ECUs, the frequency drift of a CAN ECU is a valuable measure in order to anticipate the timing accuracy and in turn the overall communication performance.

For TDMA-based FlexRay communication, frequency drift of an ECU quartz leads to mismatching of assigned TDMA slots for data transmission. As depicted in figure 4.5, an ECU, e.g. the ACC radar sensor, acquires data and processes it to FlexRay messages. If the time base of the ECU is subject to frequency drift, the very next TDMA slot for transmission might be mismatched as the drift-prone data from the operating system is not at the FlexRay bus controller in time. Hence, the ECU has to wait for the next slot, causing additional delay in data transmission.

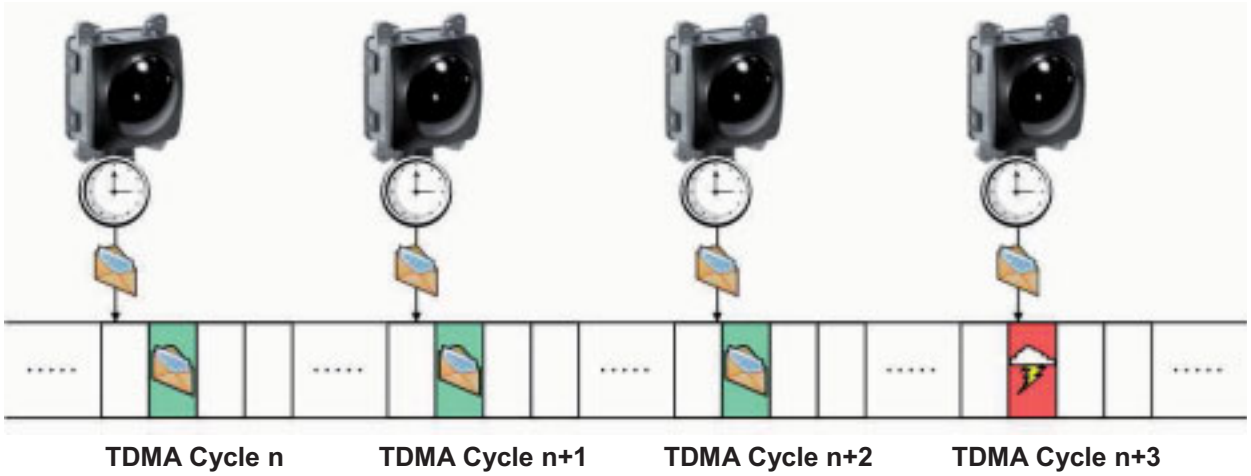


Figure 4.5: Effect of Frequency Drift of FlexRay ECU [30]

Frequency Drift of Data Logging Device

First and foremost, the frequency drift of the CONDALO data logger was determined, as the timing information available from the assigned timestamps serves as reference time base for all following investigations. Of course, the controller quartz of the data logger is subject to frequency drift, too.
To derive the drift of the controller, an existing measurement setup at the computer laboratory of the *Chair of Computer Science 7 at the University Erlangen-Nürnberg* was employed. The basic infrastructure is depicted in figure 4.6. A receiver for the *Global Positioning System (GPS)* on the rooftop of the Computer Science Building generates a digital impulse exactly every second. This impulse is recorded via the digital line-in port of the data logger, which was configured such that a timestamp is assigned to each rising edge of an GPS impulse.

4114 GPS impulses were recorded, complying to 4114 seconds of measurement duration. The timestamp for the first impulse recorded by the data logger was at 0.588856 s (*start time*), the timestamp for the last recorded impulse at time 4114.619792 s (*end time*). This yields an effective measurement time in the data logger of 4114.030936 s (*actual time*) compared to the reference time of 4114 s (*target time*) and thus a frequency drift of +7.519688 ppm, which is computed according to the following formula:

$$drift = \frac{(actual\_time - target\_time)}{(target\_time)} \times 10^6,\ (ppm)$$

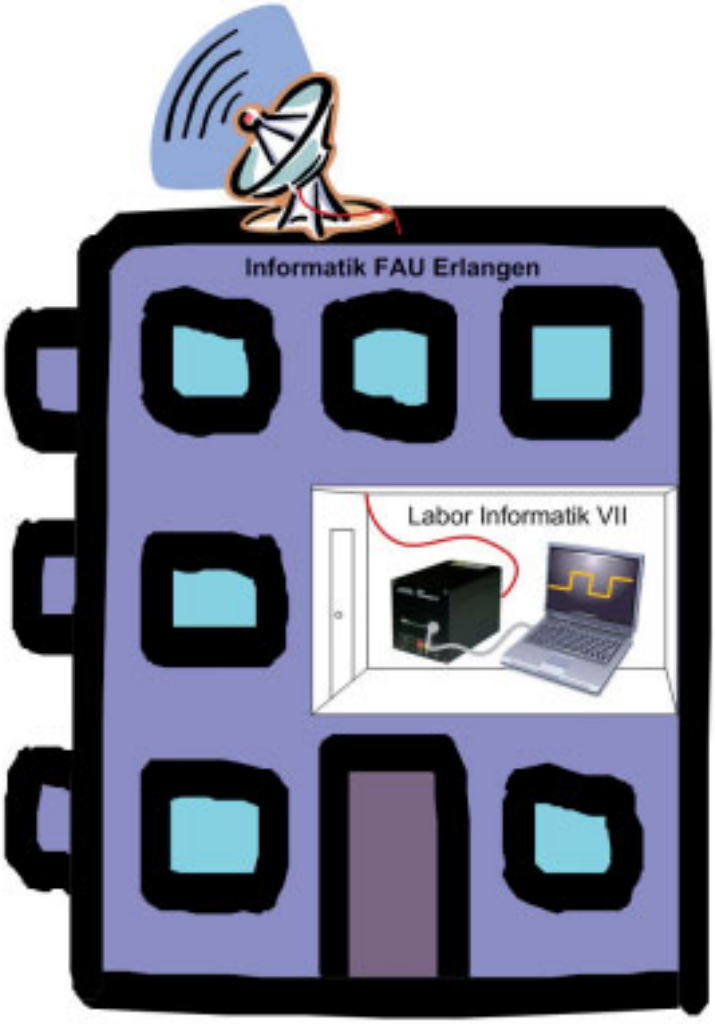


Figure 4.6: Measurement Setup for Evaluation of Data Logger Drift [30]

In detail, a frequency drift of +7.519688 ppm resembles a positive deviation of 7.519688 μs in the time span of exactly one second. The clock of the data logger shows the time 1.000007519688 s at the GPS time 1.000000000000 s, which means the quartz of the data logger is oscillating *faster* than the reference time. In figure 4.7, the plot of the timing error, resulting from frequency drift, evolving over the measurement time is depicted. The x-axis denotes the measurement time from start to end. On the y-axis, the drift of the data logger, i.e. the increasing deviation of the data logger's timestamps from the GPS reference time, is shown.

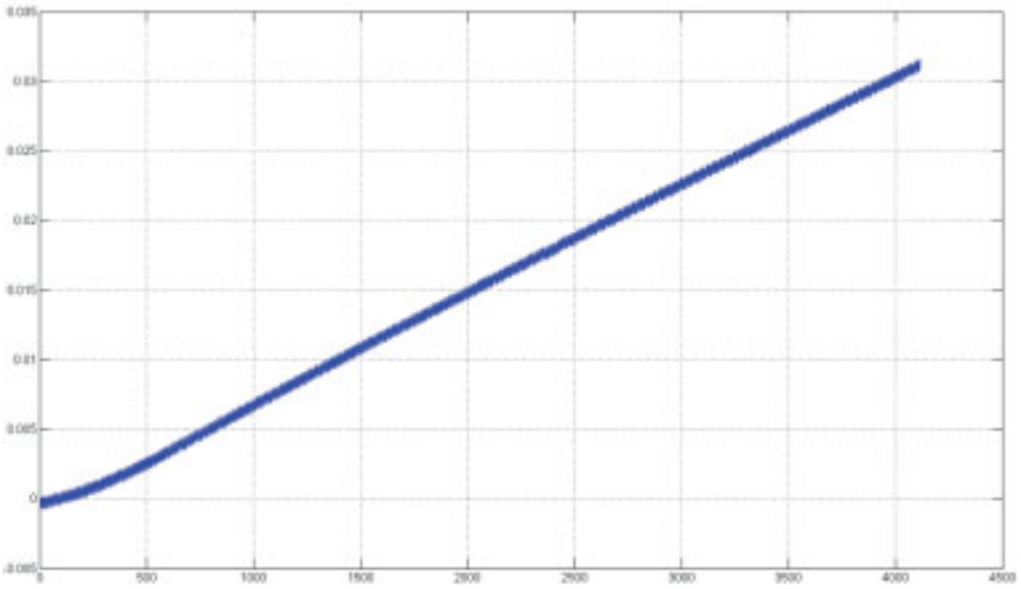


Figure 4.7: Timing Error of Data Logger

Frequency Drift of ECUs

Knowing the frequency drift of the device which is capable of recording data and assigning timestamps for communication – the CONDALO CCO DLIII data logger – the individual frequency drifts of ECUs could be determined. To this end, the recorded cyclic in-car communication data was evaluated using CAPL and CANoe. The statechart depicted in figure 4.8 resembles the functionality of the CAPL routine for determination of individual ECU frequency drifts.
Every incoming CAN message is inspected and it is determined whether it is the first message recorded of a particular type. If so, the CAN identifier and the expected cycle time are written to an external file. The cycle time of each message is available from a communication database. Subsequently, and also for every message of same type arriving consecutively to the first message, the timestamp of data recording is determined and an integer-valued receive counter is incremented. Figure 4.9 shows a screenshot of an output file as generated by the CAPL routine for evaluation of recorded communication data. The receive counter, the timestamp of reception, the time difference to the last received message of same type and the CAN identifier are written to the file. When the measurement stops, a summary is generated, containing start time and stop time of the measurement, from which the elapsed time in the data logger is computed. The number of recorded messages times the dedicated cycle time yields the target time span

which should have elapsed if no frequency drift is present at the data logger and the ECUs. The elapsed time in the data logger and the expectable target time allow for a determination of the timing error of the measurement. In the example the outcome is +0.027389 s, which means that *more* time has elapsed than expected. From this information, the frequency drift of the ECU sending the recorded message can be determined directly. For some exemplary ECUs, table 4.1 provides timing information for the actual measurements and the resulting frequency drift in the rightmost column.

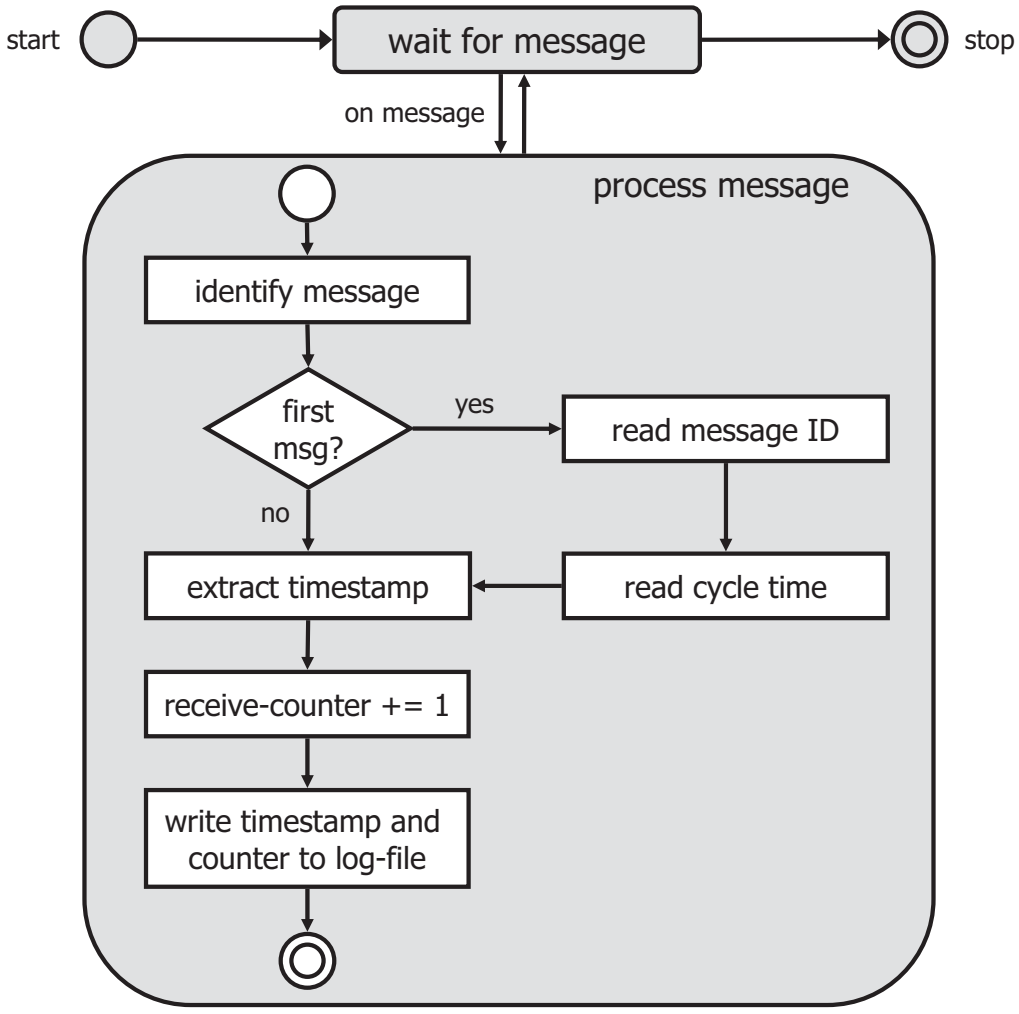


Figure 4.8: Statechart for Determination of ECU Frequency Drift

ecudrift_summary.txt - Editor

Datei Bearbeiten Format Ansicht ?

```
date Mit Nov 19 14:14:15 2008
base hex timestamps absolute
not internal events logged
// version 7.0.1


****************************************
* Evaluation of ECU Frequency Drift *
****************************************

-------------------------------------------------------------------------------------------------
        Message #       |       Timestamp       |       Time-Difference         |       Message ID
-------------------------------------------------------------------------------------------------
                0       |       12090.011752    |       12090.011752            |       0x040
                1       |       12090.061235    |           0.049483            |       0x040
                2       |       12090.111158    |           0.049923            |       0x040
                3       |       12090.161229    |           0.050071            |       0x040
                4       |       12090.211438    |           0.050209            |       0x040
                5       |       12090.261226    |           0.049788            |       0x040
                6       |       12090.311369    |           0.050143            |       0x040
                .       |             .         |             .                 |         .
                .       |             .         |             .                 |         .
                .       |             .         |             .                 |         .
            41995       |       14189.788814    |           0.050042            |       0x040
            41996       |       14189.839362    |           0.050548            |       0x040
            41997       |       14189.888775    |           0.049413            |       0x040
            41998       |       14189.938721    |           0.049946            |       0x040
            41999       |       14189.989141    |           0.050420            |       0x040

Summary:
========
Start Time of Measurements:     12090.011752 s
End Time of Measurements:       14189.989141 s
Elapsed Time in CONDALO:         2099.977389 s
Number of Messages:                    41999
Message Cycle Time:                    0.050 s
Resulting Target Time:           2099.950000 s
Timing Error of Measurements:      +0.027389 s
```

Figure 4.9: CAPL Output File for ECU Frequency Drift Evaluation

Table 4.1: Frequency Drift Evaluation for Exemplary ECUs

ECU	Start Time (s)	Stop Time (s)	Actual Time (s)	Target Time (s)	Drift (ppm)
CCO DLIII	*0.588856*	*4114.619792*	*4114.030936*	*4114.000000*	*+7.519688*
Airbag	12090.011752	14189.989141	2099.977389	2099.950000	+13.042691
Park. Brake	12090.007879	14189.994961	2099.987082	2095.200000	+2284.785223
ESP	12090.008347	14189.996895	2099.988548	2100.110000	-57.831264
Gearbox	12090.000265	14189.997266	2099.997001	2099.980000	+8.095791
El. Steering	12090.001644	14189.997971	2099.996327	2100.180000	-87.455837
Engine	12090.005710	14189.994234	2099.988524	2099.940000	+23.107327
Vision	12090.028705	14189.931174	2099.902469	2100.160000	-122.624466
Rev. Belt L	12090.035719	14189.919465	2099.883746	2099.900000	-7.740368
Rev. Belt R	12090.034413	14189.919221	2099.884808	2099.900000	-7.234630
LCA	12090.018471	14189.996164	2099.977693	2099.720000	+122.727316

4.4.2 Durations and Distributions of CAN ECU Startup Times

In [29], analytical upper bounds were derived for delays in a CAN bus system with priority-based media access, under the assumption of the worst-case scenario of a bus-wide synchronous communication startup of all ECUs. The simultaneous start of data transmission yields the maximum number of collisions at media access and thus the maximum waiting times for ECUs to transmit a message.
However, in real operation, the startup phases of different ECUs might vary significantly and the analytically computed worst-case is a valid, but rarely occurring, event in daily operation of the vehicle. The duration of ECU startup phases depends on the performance of the processor, the complexity of internal software routines, the extent of internal and external diagnosis tasks, the charge of capacitors and several other aspects.
A methodic approach to obtain realistic use-case values for CAN communication startup phase durations, and in turn the implicit schedule of media access, is presented in the following.

Clamp-15 and Clamp-30 ECUs

According to the strategy of power supply and wakeup capabilities, automotive controllers can basically be divided into ECUs which are able to start operation directly from the supply voltage of the battery (*clamp-30 ECU*) and ECUs which operate only if the ignition switch of the car is activated by the ignition key (*clamp-15 ECU*).
While clamp-15 ECU startups are directly related to an engaged ignition switch, clamp-30 ECU startups involve a sophisticated network management (NM) by the gateway. This includes monitoring of timers for wakeup, sleep and *wait-bus-sleep (WBS)* modes of ECUs, sending of NM messages and several other aspects. Figure 4.10 illustrates the process of shutting down a clamp-30 ECU. When the gateway stops periodic sending of NM messages, the ECUs change to the sleep mode if during a two-phase timeout period no further NM messages are received. A meaningful detection of clamp-30 startups requires a detailed inspection of the NM message content and the respective states of timers and timeouts.

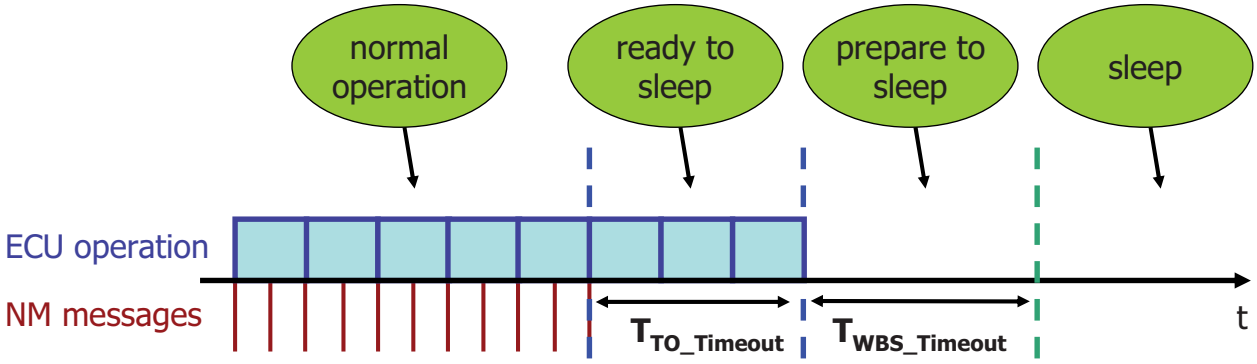


Figure 4.10: Change to Sleep Mode of Clamp-30 ECU

Data Evaluation Strategy

Knowing the startup phases, it is possible to evaluate the behavior of individual ECUs regarding the duration from activating the controller (either by clamp-15 or clamp-30) until the ECU is ready to communicate with other ECUs. In addition, the single controller startup phases add up to a general view on the CAN bus system with respect to the probability of simultaneous communication startups and collisions at media access.
In order to obtain data samples for both clamp-15 ECUs and clamp-30 ECUs, it was made use of the network management information as transmitted by the gateway for clamp-30 ECUs and of the analogue signal representing an active clamp-15 at an engaged ignition switch. The statechart in figure 4.11 depicts the logical structure of the implemented CAPL routines. Basically, it is waited for any CAN message on the bus when starting the measurement. In the *process_message* state, it is determined for an incoming message whether it was sent by a clamp-15 ECU or a clamp-30 ECU. In case of an active clamp-15, the wakeup can directly be computed by tracking the timestamp of the first message received from the respective ECU. In case of clamp-30, the actual state of the network management must be incorporated, which means checking for any non-expired wakeup- or sleep-timeouts, setting of new timers, or logging the time instance of a CAN messages from a clamp-30 ECU. The process is automated in a sense, as it examines the whole CAN bus data traffic until for each ECU, regardless of its power supply strategy, a first CAN data message was detected and the actual duration of the controller startup phase could be determined.

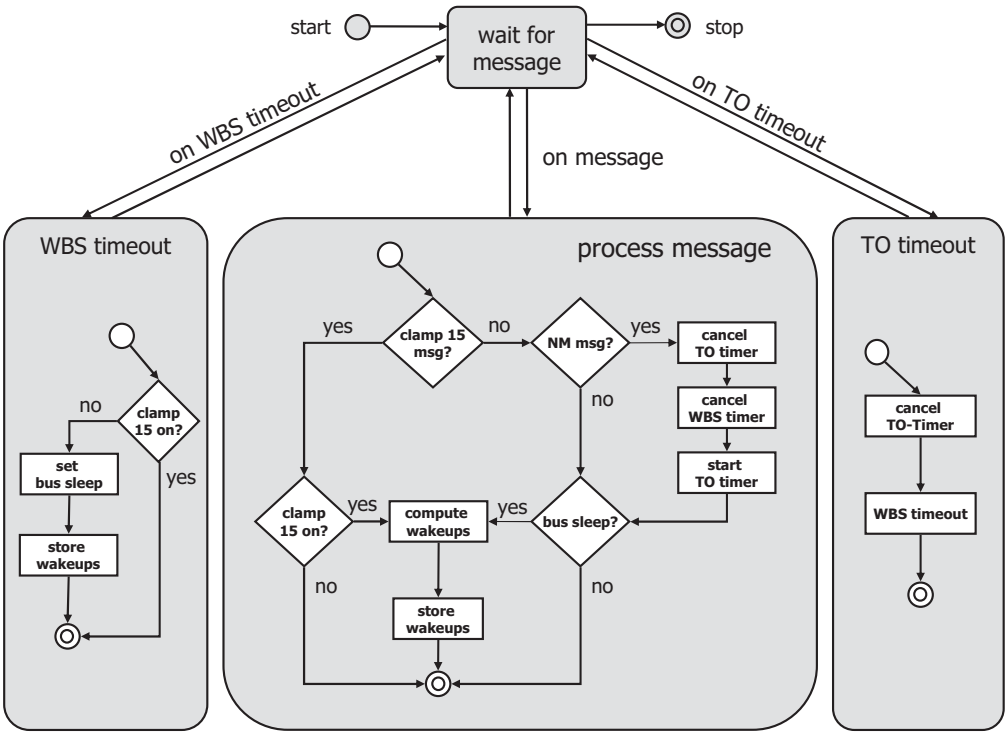


Figure 4.11: Statechart for Determination of ECU Startup Durations

Measurement Data Analyses

A total of 221 prototype measurement cycles were conducted and 2358 data samples for durations of ECU communication startup phases were obtained for the 22 active CAN ECUs.
The startup duration is defined as the time span from initiation of in-car communication to the point of time, when the respective controller booted up completely and sent a first CAN frame. For clamp-30, the start of communication is initiated by NM from the gateway, e.g. waking up the door control ECU when the central locking receives an *open* signal from the driver's remote control. For clamp-15, communication starts by manual engaging of the ignition switch. For some exemplary ECUs, values for minimum, mean and maximum startup durations are listed in table 4.2.

An ECU is expected to comply with certain constraints of communication. For example, other ECUs expect to receive CAN data before a maximum time after activation of clamp-15 or clamp-30 has passed. To assess, whether any ECU violates such bus-wide communication dependencies after startup, the startup time

Table 4.2: Communication Startup Durations for ECUs

ECU	Clamp	Startup Duration (ms) Minimum	Mean	Maximum
Airbag	15	112	122	131
Body Computer	30	9	23	73
Engine	15	39	74	82
Door Control	30	11	21	102
ESP	15	19	62	79
Rev. Belt	15	162	164	167
LCA	15	247	250	252
Gearbox	30	121	157	167

durations as denoted in table 4.2 are valuable information about communication behavior.

Keeping in mind the priority-based CAN media access, the results from table 4.2 can also be used to derive an estimation, whether two or more ECUs typically wakeup close to each other or whether intervals of individual wakeup times are likely to overlap, yielding increased probabilities for collisions in these time frames. If critical constellations can be identified, the gateway might engage in the startup process of clamp-30 ECUs via network management. For clamp-15 ECUs, the startup durations might be adapted – speeded up or delayed – internally before startup.

The type of the distribution of measured startup durations, as well as relevant distribution-specific parameters, were evaluated using ExpertFit®. Statistical evaluation was performed for all ECU startup times which were measured, exemplary results and graphics are provided for the *Airbag ECU (Airbag)*, the *Lane Change Assist ECU (LCA)* and the *Reversible Belt Tensioner ECU (RBT)*. The probability distribution functions that fit best to the measured data samples are a Log-Logistic distribution for the Airbag, a Weibull distribution for the startup behavior of the LCA and a Random Walk distribution for the RBT, respectively.

Table 4.3 lists all relevant distribution parameters as computed by ExpertFit®. For the Airbag, 316 measurements from the real system were available. Note that the number of 221 observations for the LCA complies with the number of conducted prototype measurement cycles for this particular ECU as well as the RBT. As the RBT is installed two times in the vehicle - for the front seats on the left and on

Table 4.3: ExpertFit® Evaluation of ECU Startup Durations

Parameter	Log-Logistic (Airbag)	Weibull (LCA)	Random Walk (RBT)
Observations	316	221	442
Minimum (ms)	120.06	247.50	161.76
Maximum (ms)	130.16	252.38	166.75
Mean (ms)	123.20	250.40	164.03
Median (ms)	123.19	250.48	164.05
Variance (ms)	16.4544e-7	9.48239e-7	8.13686e-7
Coeff. of Variation	1.041e-2	3.88883e-3	5.49934e-3
Skewness	0.41683	-0.56557	-0.07891

the right - 442 observations were obtained, i.e. 221 for each RBT. In addition to the minimum, mean and maximum values for the set of measured data, as already provided by table 4.2, the moments of higher order, like variance and skewness, are included in table 4.3.

In figures 4.12, 4.13 and 4.14, so-called *frequency-comparison plots* as generated by ExpertFit® are shown for evaluation of Airbag, LCA and RBT, respectively. The height of the blue bars resembles the proportion of measured data around a certain numeric value for startup duration. The red bars depict the corresponding proportion of samples from the suggested probability distribution. Hence, the smaller the deviation between a pair of blue and red bars, the better resembles the suggested distribution the actual behavior of measurement data. Obviously, the distributions resemble the nature of the real-life measurement data closely.

4.4.3 Cycle Time Jitter of CAN Messages

The priority-based CAN media access is a potential source of delay. Whenever two or more ECUs attempt the bus simultaneously, the ECUs trying to send messages with lower priorities than the currently highest one have to retreat from sending and wait for an idle bus. Generalizing from the particular case of simultaneous media access to a wide range of typical scenarios in CAN data transmission, messages of rather low CAN priorities should more often be prone to lose bus arbitration than high priority CAN messages. To reveal this effect, investigating

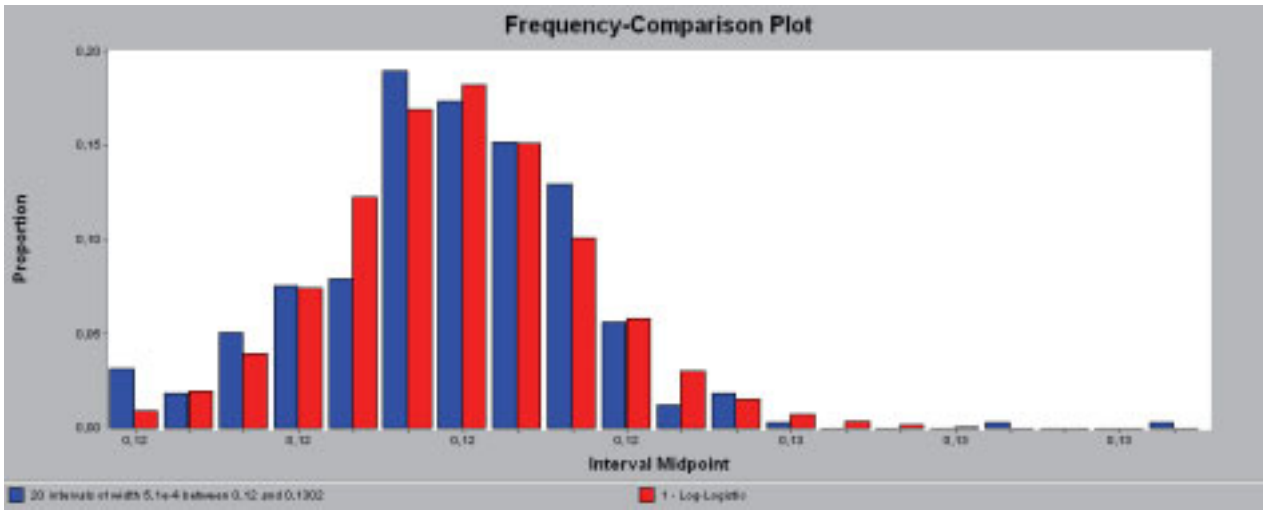


Figure 4.12: Measurement Data Analysis (Airbag)

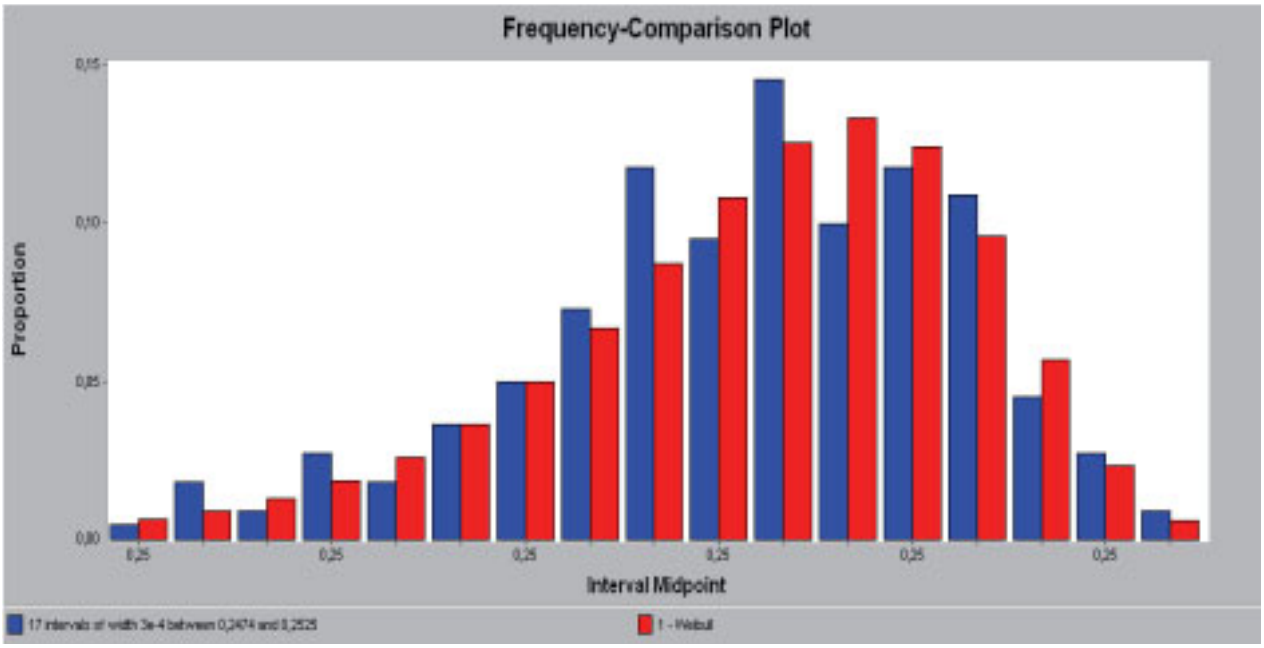


Figure 4.13: Measurement Data Analysis (LCA)

the transmission timing of cyclic CAN messages of different priorities, sent from one ECU, is a promising approach for several reasons:

- Cyclic sent messages *ideally* appear on the bus according to the sending cycle, i.e. the interarrival time between two consecutive instances of messages of a certain CAN priority is equal to the cycle time.

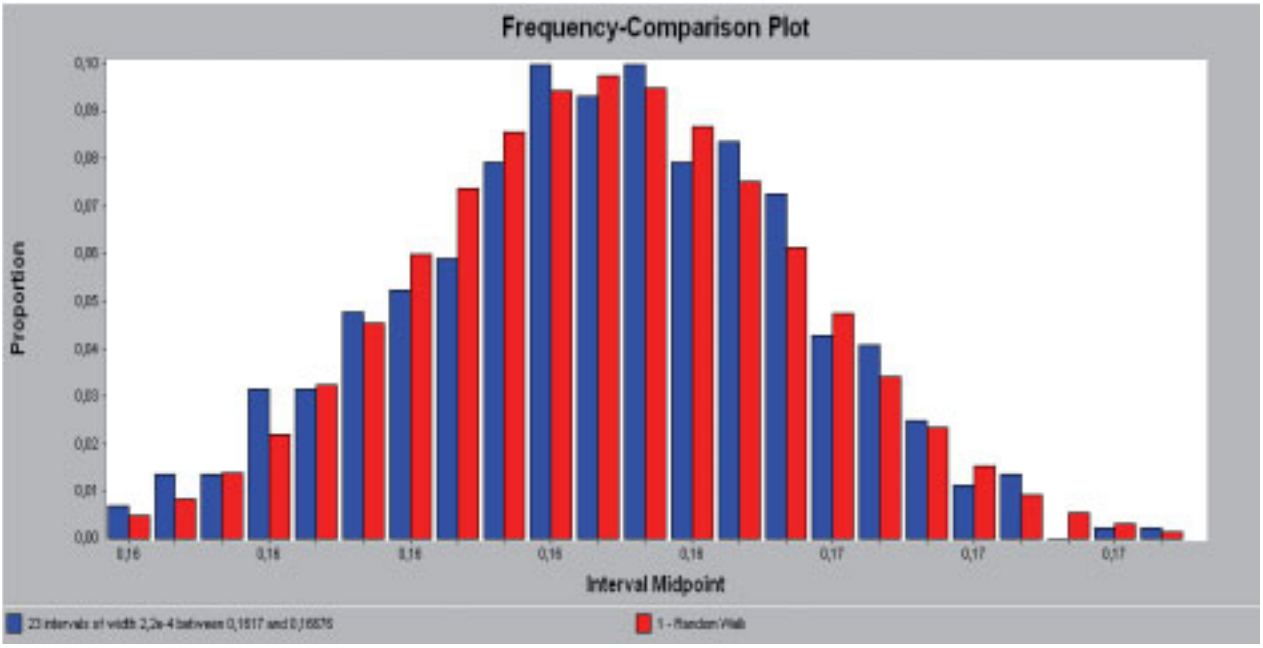


Figure 4.14: Measurement Data Analysis (RBT)

- Any deviation in interarrival times is either caused by an inaccurate controller timing, the frequency drift, or by varying delays at media access, i.e. the actual time, when the next message of same type can be transmitted on the bus.
- As stated beforehand, different ECUs exhibit differences in frequency drift. Thus, interarrival times of messages from various ECUs cannot be compared directly to figure out CSMA/BA effects. However, messages from the same ECU experience the same inaccuracy of the controller quartz. Hence, the frequency drift can be seen as a constant factor and these messages can be compared directly.
- Consequently, any deviation in cycle times for messages from one ECU must be caused by a varying delay at CAN media access.
- Messages of higher (or highest) CAN priority are expected to show minor deviations – jitter – in cycle times, as the ECU should be able to transmit them directly, except for at most one period of non-preemption.
- Messages of lower (or lowest) CAN priority are more likely to be subject to significant deviations in interarrival times. Besides direct transmission to an idle bus – resembling exactly the cycle time between to consecutive

messages – constellations can arise, where several higher priority messages are transmitted before. This prolongates the interarrival time and yields a positive jitter, whereas a shorter time gap to the appearance of the next message, e.g. if the bus is by then idle again, leads to a negative jitter.

These assumptions could be validated by measurements from the real-life prototype vehicle. The histograms in figures 4.15 and 4.16 depict the distributions of interarrival times for a high priority CAN message with CAN-ID 64 and 50 ms cycle time and a low priority CAN message with CAN-ID 1408 and a cycle time of 200 ms, respectively. Both messages are sent by the same ECU. The measured interarrival times are depicted in seconds on the x-axis. The values were obtained by evaluating the timestamps of each two consecutively sent messages of same type. Determining the time span between these messages yields the effective message cycle or interarrival time. The number of measured values for each cycle time along the x-axis is depicted on the y-axis.
Obviously, the histogram in figure 4.15 is significantly narrower than the one in figure 4.16. The concentration of values around the cycle time of 50 ms for the high priority CAN message indicates less constellations with non-performable direct bus access. The histogram for the low priority CAN message shows a wider range of measured interarrival times. These deviations strengthen the assumption that a rather low priority CAN frame will considerably more often be subject to a loss of media access than a high priority CAN message. This is emphasized by incorporating the standard deviation of cycle times for both sets of measured data samples, which is 0.121203 ms for the high priority CAN message and with 0.332465 ms almost three times higher for the low priority CAN message.

4.4.4 Routing Delay in Central Gateway

In an in-car communication system composed of several bus systems and tens of electronic devices, the dependencies in data exchange between various ECUs are likely to span across the borders of local buses. As in any other communication network, a gateway serves as central node, which is capable of interconnecting the various, technologically different bus segments and of routing of messages.
For many networked applications, the real-time capabilities of the gateway in terms of routing delays and timeliness of data transfer are a crucial factor for the overall function performance. As motivated for time-critical precrash data transfer

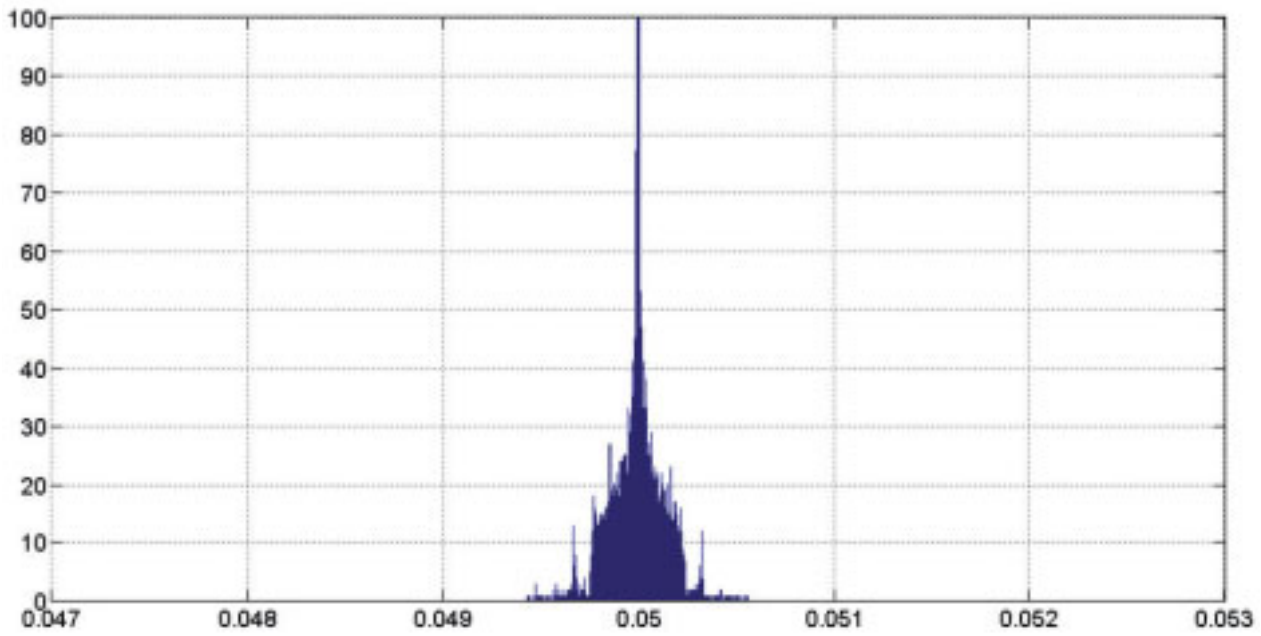


Figure 4.15: Distribution of Cycle Times for High Priority CAN Message

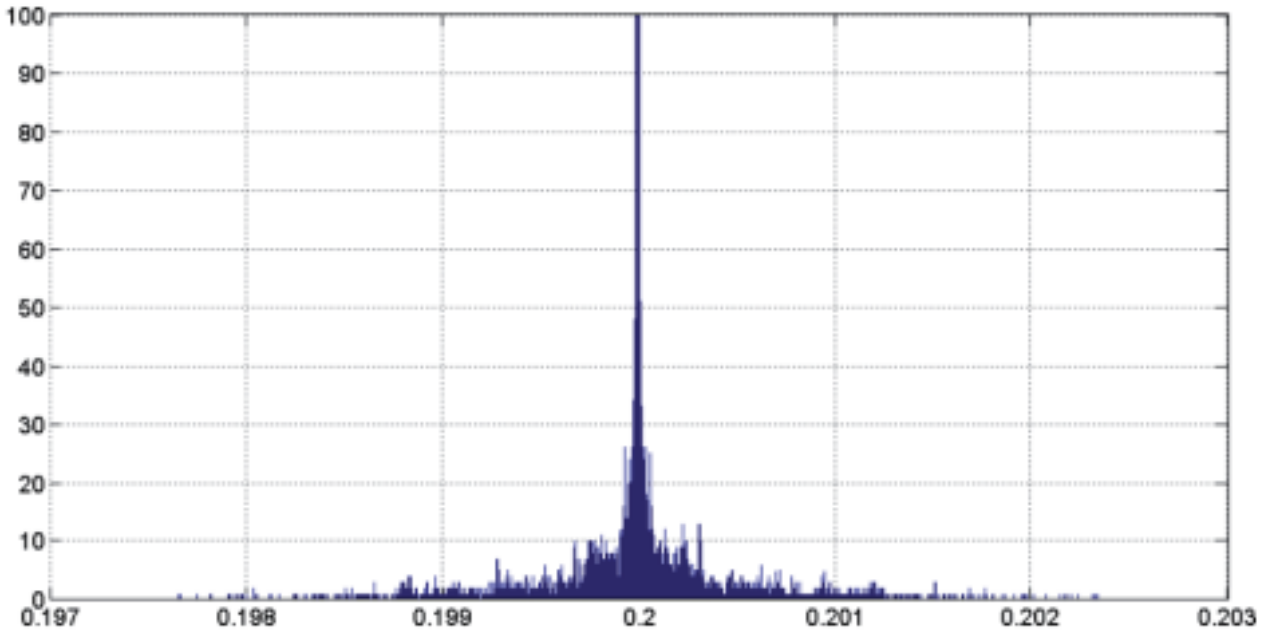


Figure 4.16: Distribution of Cycle Times for Low Priority CAN Message

in chapter 1, the delay between sensing a dangerous situation, sending the information and receiving and appropriate reacting to the threat strongly determines the efficiency of any precrash means. As the following investigations clearly point out, routing of data is a potential bottleneck in an end-to-end communication scenario of networked ECUs and distributed applications.

Routing Dependencies and Communication Parameters

From the 5 CAN bus segments and the FlexRay bus in the prototype vehicle, bidirectional routing dependencies were determined between 5 of the overall 28 ECUs on the buses CAN_1, CAN_2 and FlexRay. Namely, the ECUs which are assumed to exchange data for safety-relevant applications are: *Engine ECU (Engine)*, *Airbag ECU (Airbag)*, *Lane Change Assist ECU (LCA)*, *Adaptive Cruise Control ECU (ACC)* and *Electronic Stability Program ECU (ESP)*. Figure 4.17 shows the respective parts of the topology of the in-car communication system.

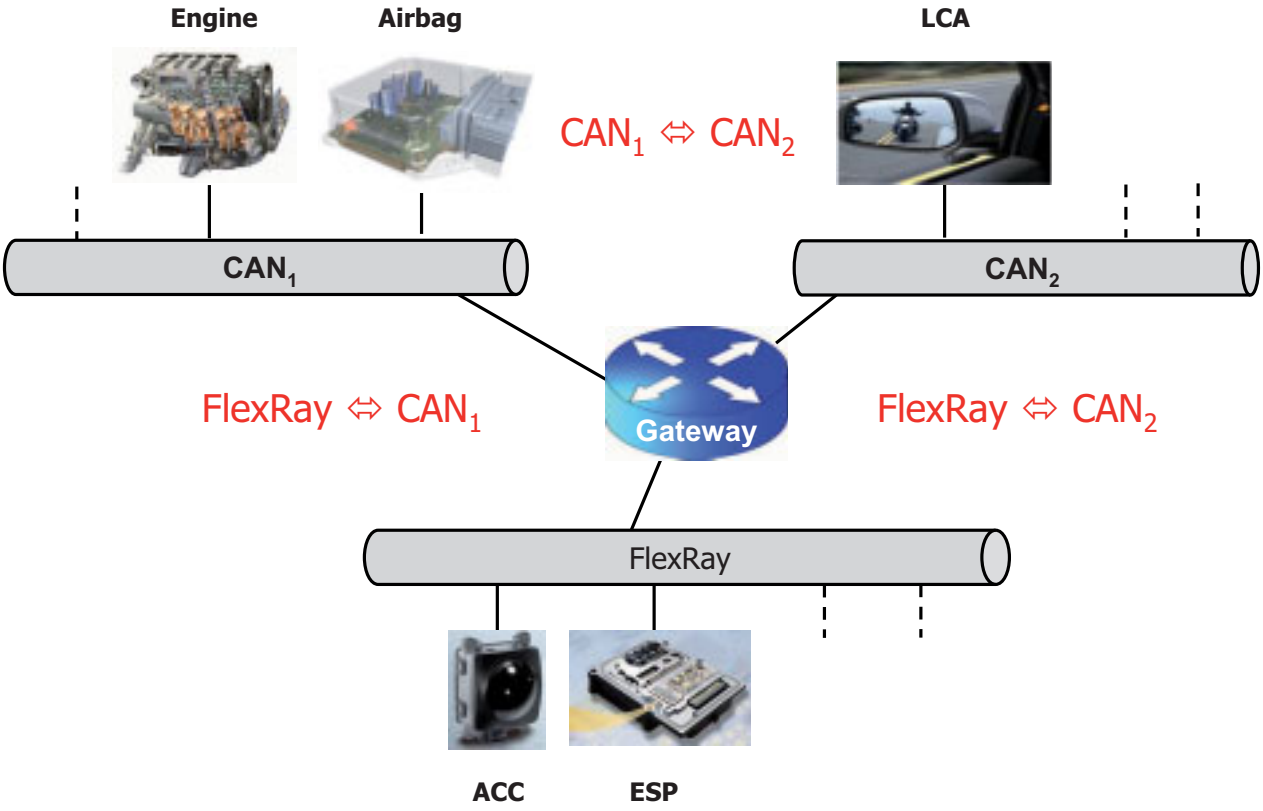


Figure 4.17: Safety-Relevant Data Exchange between ECUs in the Network

Obviously, routing is performed by the gateway for three different constellations: from CAN to CAN (1), from CAN to FlexRay (2) and from FlexRay to CAN (3). The aspects which are expected to be relevant for the routing delay are the waiting time for CSMA/BA media access for a CAN message of certain priority

Table 4.4: Communication Parameters of Safety-Relevant Messages

Name	#msg	CAN ID	CAN Cycle (ms)	FlexRay Cycle (ms)
ACC_01	26897	803	40	40
Airbag_01	21564	64	50	5
Airbag_02	5393	1312	200	–
ESP_01	53834	256	20	20
ESP_02	26919	776	40	40
Engine_01	107980	128	10	5
Engine_02	107982	261	10	5
LCA_01	26903	783	40	20
LCA_02	26907	788	40	20
LCA_03	26903	802	40	–

in (1) and (3) and the delay from the point of time when data is ready to be sent to FlexRay by the gateway until the next dedicated time slot begins in (2). Several messages were picked out for a determination of routing delay. Table 4.4 presents the message parameters and has to be read as follows:

- *Name* is the descriptive name of the message, composed of the sending ECU plus a unique number for each message.
- *#msg* is the number of messages recorded in order to determine the routing delay.
- *CAN ID* is the CAN identifier of a message which is sent from or to a CAN bus.
- *CAN Cycle (ms)* is the cycle in which the message is periodically sent on a CAN bus.
- *FlexRay Cycle (ms)* is the time between consecutive TDMA time slots to send the message.

The CAN-IDs are network-wide assigned uniquely to a message, a message from a CAN bus retains its ID on every other CAN segment and a message from FlexRay is assigned the same CAN-ID for every CAN bus it is routed to. If a message is routed from FlexRay to a CAN bus, the cycle in which the message appears on

the sink bus is determined by the TDMA cycle time, as the gateway just tries to forward the message as it receives it. The other way round, the TDMA cycle time for a message which is routed from a CAN bus to FlexRay is typically chosen such that the waiting time for the gateway's next TDMA time slot is as short as possible. One practical approach is to chose half of the initial CAN cycle[1], e.g. as done for the messages `Engine_01`, `Engine_02`, `LCA_01` and `LCA_02`. If a message contains profoundly time- and safety-critical information, even shorter TDMA cycle times can be chosen to minimize the routing delay, e.g. as done for `Airbag_01` with a 5 ms FlexRay cycle.

Data Evaluation Strategy

The delay of a message was determined according to the evaluation strategy as described by the statechart in figure 4.18.

The basic information is derived from the timestamps the data logger assigns to each recorded message. A particular message, which is routed from a source to a sink bus, will appear two times in the log-files of the data logger. First, when it is transmitted from the source bus to the gateway and second, when the gateway transmits the message to the destination bus. Each of the recorded instances of the message is provided with a timestamp. This allows to determine the routing delay directly by subtracting the time for sending the message on the source bus from the time for transmission from the gateway to the sink bus. As a message can not arrive on the destination bus before it was sent on the source bus, this approach yields reasonable results for any source/sink-pair of consecutively appearing messages.

Measurement Data Analyses

Various important aspects of routing in automotive ECU networks are emphasized by the results in table 4.5. The mean and maximum routing delays are denoted in a column-wise style according to "source $\Rightarrow$ sink". An entry of "-" indicates that

[1] If the FlexRay cycle is 1/n-th of the CAN cycle, the same message is sent n times from the gateway to the sink bus. The first time it is sent, a bit indicates new data content, the following $n-1$ times, the gateway sets the bit such that it indicates repeated data.

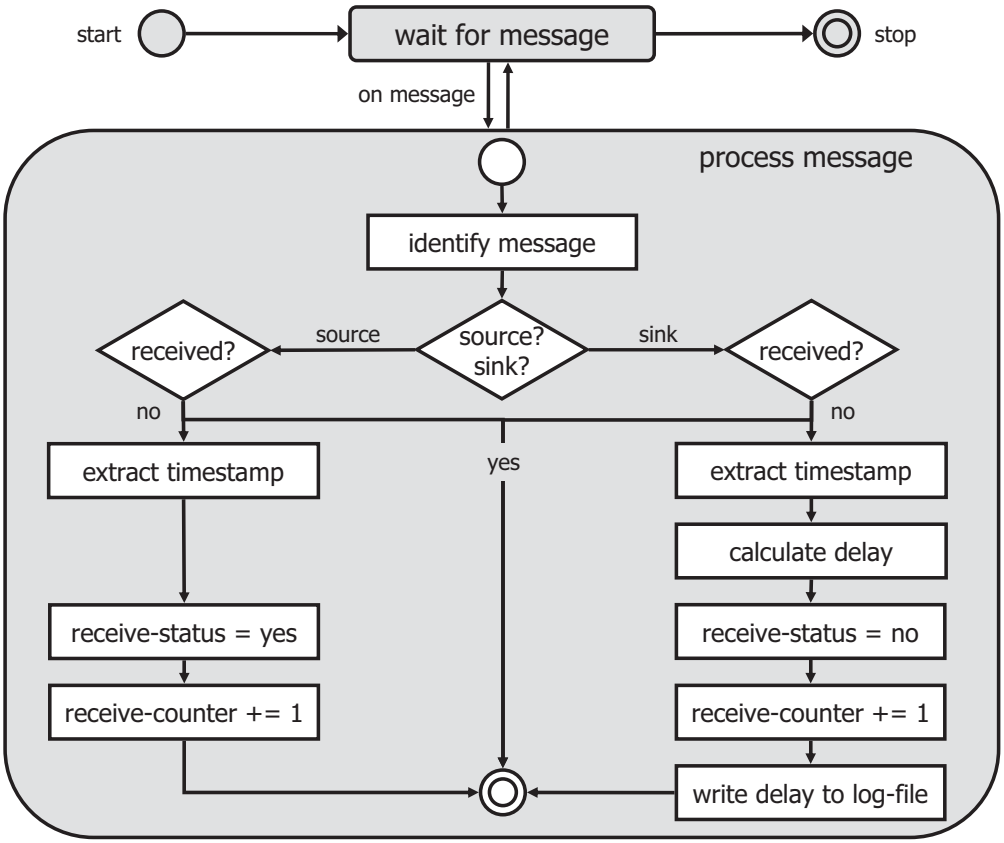


Figure 4.18: Statechart for Routing Delay Evaluation

the message is not routed in the scenario described by the column. Depending on the busload of the sink CAN bus and the number of messages with higher priority, the CSMA/BA media access might be significantly delayed if the message has a rather low priority. For example, routing the message `Airbag_02` with CAN-ID 1312 from CAN_1 to CAN_2 takes an average/maximum time of 4.56 ms/11.01 ms, whereas routing the message `Airbag_01` with CAN-ID 64 takes only 0.42 ms/1.16 ms on average/maximum. The same holds for routing the messages `ACC_01` and `ESP_01` from FlexRay to CAN_1, with average/maximum delays of 1.37 ms/3.50 ms and 1.34 ms/2.65 ms, respectively.
Another considerable issue is the delay a message might experience at routing to the TDMA-based FlexRay. An ECU is only allowed to access the bus in its dedicated time slot in the static segment of a FlexRay cycle. If the data to be sent is available when the time slot begins, it can be transmitted straight in the current time slot. Otherwise, the ECU has to wait for the next time slot, which might take as far as one cycle time. Actually, this leads to a remarkable delay in message transfer. This effect becomes obvious for the messages `Airbag_01`,

`Engine_01`, `Engine_02`, `LCA_01` and `LCA_02`, where the routing delay from the buses CAN_1 and CAN_2 to FlexRay is in the range from half a TDMA cycle time on average, up to a complete TDMA cycle for maximum routing delay. As the efficiency of time- and safety-critical applications might significantly decrease within a few milliseconds, such routing delays can be of considerable impact on the overall application performance.

Table 4.5: Delays for Routing of Safety-Relevant Messages

Message	Mean/Maximum Routing Delays (ms)					
	$CAN_1 \Downarrow CAN_2$	$CAN_2 \Downarrow CAN_1$	FlexRay $\Downarrow CAN_1$	$CAN_1 \Downarrow$ FlexRay	FlexRay $\Downarrow CAN_2$	$CAN_2 \Downarrow$ FlexRay
ACC_01	–	–	1.37/3.50	–	1.27/3.33	–
Airbag_01	0.42/1.16	–	–	4.49/5.00	–	–
Airbag_02	4.56/11.01	–	–	–	–	–
ESP_01	–	–	1.34/2.65	–	1.30/1.88	–
ESP_02	–	–	–	–	1.34/2.29	–
Engine_01	–	–	–	3.45/5.00	–	–
Engine_02	0.41/2.07	–	–	3.42/5.00	–	–
LCA_01	–	–	–	–	–	10.60/20.00
LCA_02	–	–	–	–	–	10.07/20.00
LCA_03	–	0.53/3.11	–	–	–	–

4.5 Discussion of Prototype Measurements and Data Evaluation Results

The prototype measurements as presented in this chapter are an important part of performance evaluation of up-to-date and future in-car data transmission. The measurement infrastructure is composed of both commonly employed hardware devices and widespread automotive software tools. Thus, researchers and responsible engineers are provided with the ability to make use of well-known and easily adaptable technologies for investigations on a broad variety of aspects and "hot topics" in future networked safety systems. To assure this, hardware configuration and software parameterization are kept modular, maintainable and intuitive, which allows for an straightforward employment of the measurement infrastructure in various other prototype vehicle configurations.
The actual choice of performance measures was strongly influenced by the real-time demands of networked safety applications. Both the ECUs incorporated in these distributed safety functions and the underlying communication infrastructure contribute to the performance of time-critical data transfer and in turn to the overall application performance. ECU-dependent performance measures – startup duration and controller frequency drift – were derived from the recorded communication data as well as measures which are more related to the network topology, namely cycle time jitter and routing delay.
As a matter of fact, the single measures cannot be seen isolated from other outcomes, moreover, there are interdependencies as the effects are influencing each other. For example, from ECU startup durations constellations might emerge with considerable CSMA/BA collisions at media access, which is in turn leading to increased cycle time jitter at ECUs or significant routing delay at the gateway, especially for lower CAN priorities.
The prototype measurements have shown that effects, which are discussed theoretically, are really physically present in the communication network of a modern vehicle. As a summary, the following peculiarities and pitfalls in in-car ECU communication could be revealed:

- Frequency drift in the ECU controller quartzes varies significantly between different ECUs. In the long-run of system operation, this inaccuracy in controller timing leads to undesired side-effects in cyclic communication,

e.g. mismatching of FlexRay TDMA slots or burst-like concentration of CAN media accesses and, as a consequence, increased CSMA/BA delays.

- The startup durations of CAN ECUs are also subject to considerable variation from one ECU to the other. At first glance, this weakens the worst-case assumption of a bus-wide simultaneous CAN ECU transmission start for real-life operation. However, several other problems might arise from actual startup durations, like communication timeouts due to excessive controller booting periods or unpredictable, significantly varying startup time spans.
- The cycle time jitter is an important indicator for losses in CAN media access. Incorporating messages from one ECU, experiencing the same frequency drift, allows for a straightforward and reasonable identification of the influence of the CAN-ID on priority-based media access.
- The gateway is the central node in the communication system. Depending on the priority of data routed to a CAN bus or depending on the TDMA cycle time of messages to FlexRay, the routing delays vary significantly. The evaluation results show that the delays for routing of a low priority CAN message may be about a factor of ten times higher than for high priority data. For FlexRay, the common-sense assumption of a waiting time of half a TDMA cycle in the mean and a complete cycle time at maximum could be validated.

Finally, the performed prototype measurements allow for determination of several system parameters, which are either inaccurately specified or completely unavailable from technical specifications of ECUs. The frequency drift of the controller quartz and the actual ECU startup durations are important measures in a system of distributed safety applications. As shown in the following, these measured data samples are valuable input parameters for simulation of the in-car communication system. They strongly contribute to a realistic modeling of the stochastic and dynamic behavior of an ECU network.

5 Discrete Event Simulation of In-Car Data Transmission

In this chapter, an approach is presented to investigate the so-called *use-case*, which resembles the typical real-life behavior of an automotive in-car communication system. By means of statechart modeling and discrete event simulation, the dynamic behavior of the network can be captured appropriately. Research results and application studies were published in [21] and [20].

5.1 Motivation

Just as in various other application domains of networking, automotive communication systems in modern cars are permanently increasing in both size and complexity. Thus, performance evaluation of such a complicated network of tens of ECUs and distributed applications becomes an inevitable task of system design. As shown in the previous chapter, prototype measurements are a reasonable way to generate important measures on the individual performance of ECUs and on the capabilities of the communication system. However, measurements require physically present systems – bus systems, ECUs and, of course, test cars. Consequently, real-life measurements and tests must be performed either late in the development process, e.g. shortly before putting a model to serial operation, or in parallel to standard development as some kind of "engineering playground" for new concepts or technologies.
A paradigm, that has been widely deployed in various domains and that allows to investigate system behavior at an early stage of the design process, is simulation. The idea of simulation is to resemble the structure and the dynamic behavior of the system, to simulate its operation and to collect meaningful performance measures during the simulation runs. Depending on the actual scope of the simulation and

the complexity of the system, the level of detail in modeling may be arbitrarily high. It can range from an almost complete resemblance of real-life instances to a very coarse modeling, restricted to basic components and behavioral patterns. Several simulation tools and libraries for well-known programing languages are available for an ease of modeling and simulation.
The main advantage of simulation is that the model is not necessarily related to the physical presence of the system which is to be simulated. This makes simulation, in most cases, cheaper, faster and less manpower-consuming than test campaigns and measurement studies. In addition, structural changes in the simulation model often require only minor, programmatic changes, compared to time-consuming reconfigurations in a real-life system.
A reasonable simulation study requires an appropriate input modeling, either as external stimuli to the system, e.g. arriving customers or data, or as stochastically captured patterns in behavior of the components, like ECU startup durations or processor failure rates. Furthermore, a simulation control should be applied to assure the statistical relevance of the outcomes by replication of single simulation runs and intensive variation of stochastical model parameters. Incorporating all this, simulation is particularly suited to generate trustworthy outcomes for dynamic system operation and to support basic design decisions at an early stage of the development process.

5.2 Discrete Event Simulation

The choice of the simulation paradigm is one of the very first steps in building a simulation model. Assuming that operation of a system is to be resembled as it evolves over time, there are two basic timing concepts possible: *continuous time simulation* or *discrete event simulation* (or, as a combination of both, so-called *hybrid simulation approaches*). In a continuous time simulation, the system's components and variables are typically described by mathematical formulas, e.g. differential equations, which are evolving continuously over time, i.e. in timesteps with a – theoretically infinitely – high granularity.
Discrete event simulation does not employ a constant advance in timing. The dynamics and actions in a model are described as *events*, for which the execution time is determined, i.e. the system time, when the event is to be performed and all system states related to the execution of this particular event are to be updated.

Consequently, the system advances not continuously in time, but from one event to the next. Typically, these events are managed in an *event-list*, depending on the actual execution times, the time spans between two events may vary as depicted in figure 5.1.

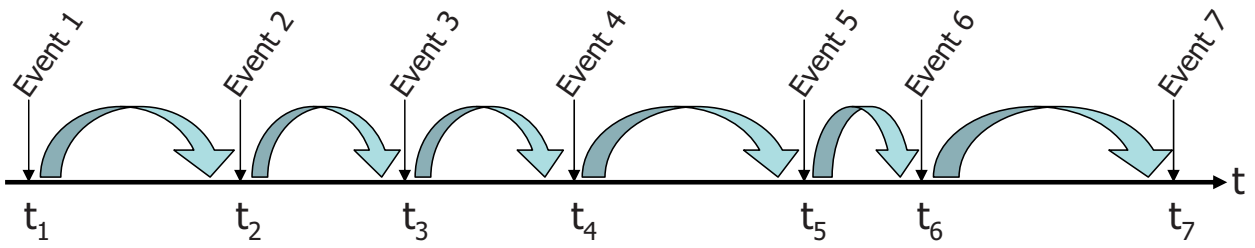


Figure 5.1: Event-based Progress in Simulation

As an example for a hybrid model containing both continuous time and discrete event simulation, just think of oil tankers arriving at a harbor for unloading of oil. The tankers arrive one after the other, thus the number of tankers at the dock changes discretely with every new arrival. To resemble this, an event list may contain the points of time when the next tanker will arrive. However, once the unloading process is started, the filling of oil within a particular tanker decreases continuously over time, which is best described by continuous time differential equations. As both paradigms, continuous and discrete, are employed in this model, it is of hybrid character [36].

5.3 AnyLogic™

The modeling tool AnyLogic™ [64] was used to build a simulation model of data transfer in the in-car communication system. AnyLogic™ supports discrete event simulation in combination with UML statecharts. The relevant system components are modeled as so-called *active objects*. These objects implement the functional and dynamic behavior of the respective instances of the real-world system. An active object is able to communicate with other objects by sending messages via ports. Figure 5.2 shows a screenshot of the AnyLogic™ modeling environment, together with a cascaded depiction of an active object, its variables, statechart and Java™ source code for behavioral description.

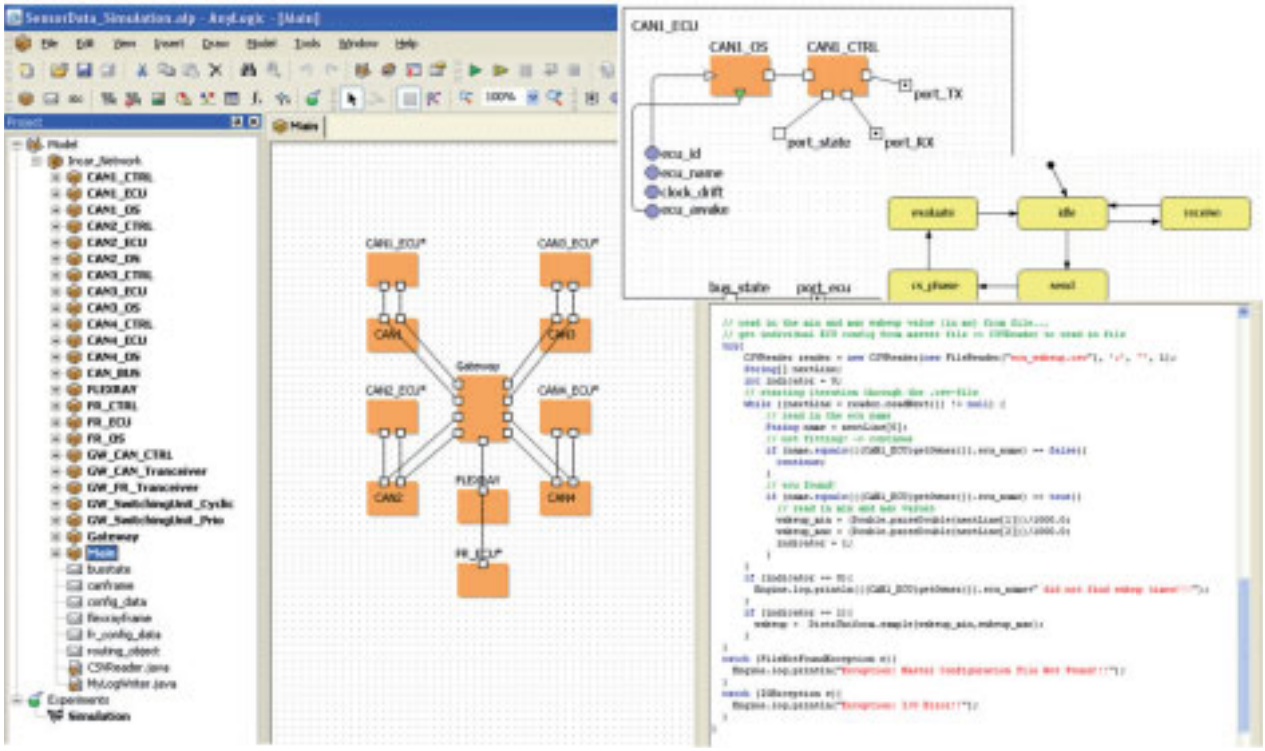


Figure 5.2: Screenshot of AnyLogic™ Modeling Environment

5.4 Modeling Elements for Simulation of In-Car Data Transmission

In this section, the modeling elements are presented, which were created in order to resemble the basic components of a modern automotive communication system, namely message objects for CAN frames and FlexRay protocol data units (PDUs), CAN bus, FlexRay and an automotive gateway.

5.4.1 Message Objects

Message objects are the central instances of port-based data exchange between active objects in the simulation model, inheriting from the Java™ class `Object`. A message object may contain several information of various data types, which are accessed from outside in the commonly known "."-notation. For example, the fictitious message object class `exampleObject` provides the following information:

- `exampleObject.name` returns a string variable bearing the name of the object.

- `exampleObject.counter` contains an integer-valued counter variable, indicating that the current message object is the n-th instance created of this particular type.

- `exampleObject.timeCreated` returns a double variable containing information on the point of system time when the actual message object was created in the simulation model.

Several types of message objects can be included in the model in parallel, each of those representing an own sub-class of `Object`. Standard or extended constructors allow for generation of new instances of the message object class according to

```
messageObject exampleObject = new messageObject()
```

Numerous messages of varying type may be "physically present" in the simulation. Messages which are no longer needed are automatically deleted by the Java™ garbage collection.
As the simulation model resembles the structure and behavior of data transmission in an in-car communication system, it appears to be worthwhile to make use of message objects to model the central data transfer instances, namely CAN frames and FlexRay PDUs.

CAN Frame

A CAN frame in the simulation is resembled by a `can_msg` message object. The message object contains more information than included in the header-, payload- and trailer-fields of a real CAN frame as presented in chapter 3. Table 5.1 lists the variables of this message object. As shown later in this chapter, the size of each CAN frame is assumed as the maximum value of 130 bits, with additional 6 bits for carrier-sense phase at CSMA/BA media access.

Table 5.1: Variables of a CAN Frame Message Object

Variable	Data Type	Description
`name`	string	Descriptive name of the message
`ID`	integer	CAN-ID of the message
`cycle`	double	CAN cycle time in milliseconds
`sender`	string	Name of ECU sending the message
`timeCreated`	double	System time of message creation
`timeAccess`	double	System time of CAN media access
`timeTransmitted`	double	System time of end-to-end transfer

FlexRay PDU

In the simulation model, the message object class `fr_pdu` is the equivalent to real-world FlexRay PDUs, in which the actual payload within a FlexRay frame is packaged. Table 5.2 provides information on the variables an `fr_pdu` contains.

Table 5.2: Variables of a FlexRay PDU Message Object

Variable	Data Type	Description
`name`	string	Descriptive name of the message
`sender`	string	Name of ECU sending the message
`length`	integer	Payload size in bytes
`timeCreated`	double	System time of message creation
`timeTransmitted`	double	System time of end-to-end transfer

5.4.2 CAN

The simulation model of the CAN communication system consists of two top-level building blocks as shown in figure 5.3, namely the `CAN ECU` and the `CAN Bus`. The stacked representation of the `CAN ECU` resembles that multiple instances of this object type may be connected to the same bus. Communication between the bus and the ECUs is performed in a twofold manner. The bus receives CAN messages from the ECUs, processes them and transmits the data as broadcast. In

addition, the bus communicates its current state – idle or busy – to each connected ECU.

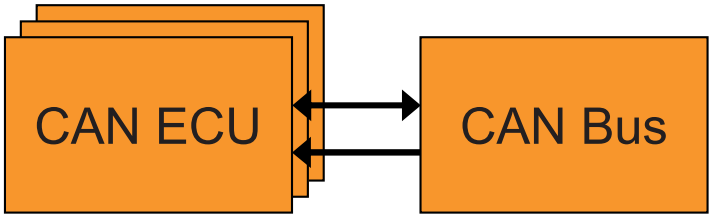


Figure 5.3: Conceptual Top-Level View on CAN Communication

CAN ECU

A CAN ECU is implemented in the simulation model as one active object on the top-level view. For an appropriate modeling of the ECU functionality, this building block is subdivided into an operating system (`ECU OS`) and a CAN communication controller (`CAN CTRL`) as depicted in figure 5.4.

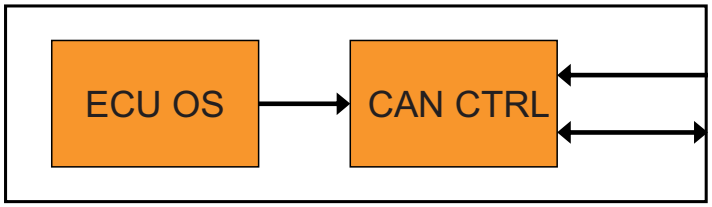


Figure 5.4: Structure of the CAN ECU Object

Operating System The real-life in-car network awakes upon an external event, like releasing door locks by remote control or starting the engine via the ignition switch. Triggered by such an event, software tasks of the operating systems are activated and the ECU starts operation. In the simulation, this behavior is captured by a transition, which performs the change from `sleep` into `awake` mode as denoted in the statechart in figure 5.5.

As validated by the prototype measurements presented in chapter 4, the duration of this startup phase varies between different ECUs and even for one ECU from one activation cycle to the next. The actual startup duration for a particular ECU in one simulation run must be determined, i.e. the timeout for the timed transition from

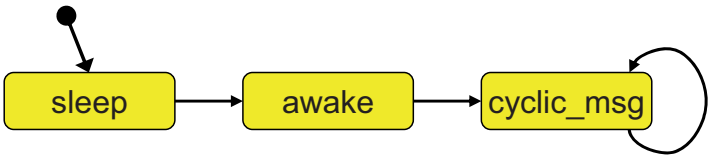


Figure 5.5: CAN ECU Operating System Statechart

state `sleep` to state `awake`. To this end, an external file, providing information on the minimum and maximum startup times for the respective ECU, is read in at simulation start and a timeout value is sampled from an appropriate distribution. Once this startup phase is over, the ECU is awake and the operating system changes to state `cyclic_msg`, where cyclic CAN messages of type `can_msg` are generated. Generation starts with an initial burst of all messages from the respective ECU. The variables for each `can_msg` object from table 5.1 are initialized accordingly. For every message, a timer from the Java™ class `CycleTimer` is set to the CAN cycle time. As the timers expire one after the other, periodical generation of messages is achieved in the following. The actual mechanism is depicted in figure 5.6, for two ECUs with dedicated durations of startup phases and cyclic CAN messages to be sent.

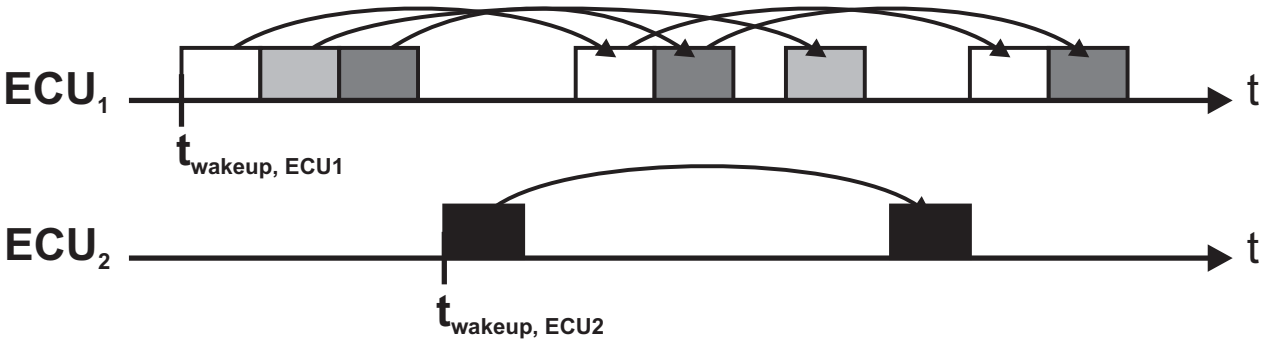


Figure 5.6: Cyclic Communication of CAN ECUs after Startup

Upon generation and initialization, the `can_msg` objects are passed from the operating system to the CAN communication controller, which is in charge of transmitting the messages to the CAN bus. This port-based message exchange is indicated by the unidirectional transition between `ECU OS` and `CAN CTRL` in figure 5.4.

CAN Controller The functionality of a CAN communication controller is implemented in the `CAN CTRL` active object as shown in the statechart in figure 5.7. A CAN controller is responsible of transmitting and receiving CAN messages, complying to the CSMA/BA media access scheme for bus arbitration and data transmission. Initially, and also if no data is to be sent or received, the controller is in the `idle` state. If no CAN message is to be sent and instead a message from any other ECU is received via the bus, the controller changes into the `receive` state and processes the incoming data. Especially, the variable `timeTransmitted` of the received CAN message object will be set to the current simulation time, indicating reception of data at the controller and hence accomplished data transfer. Each cyclic CAN message generated by the ECU operating system is stored in the controller object in a list. Messages are sorted by CAN-ID, with the lowest ID on top of the list. Therefore, a hashing-based approach for data storage is implemented in each `CAN CTRL` active object, with a sorted linked list of type `TreeMap` from the Java™ class library `java.util`.
If the CAN bus is idle and the sorted list is non-empty, the controller changes from `idle` to `send` state. It retrieves a copy of the topmost message object from the list, sets the variable `timeAccess` to the current simulation time and transmits the message. Afterwards, the controller proceeds into the `cs_phase` state, waiting for what it receives from the bus. This approach abstracts from a bit-by-bit bus arbitration mechanism, i.e. the CAN controller sends the complete message and, as shown later, the `CAN Bus` object is responsible for determining which message won the arbitration by evaluating CAN-ID and timestamp of media access as set by the controller.

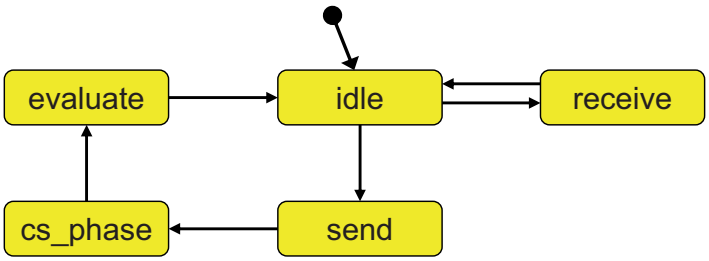


Figure 5.7: CAN Controller Statechart

After the carrier-sense phase, the bus broadcasts the message which has actually won media access. The CAN controller receives this message and changes from `cs_phase` state to `evaluate` state, in which the received data is compared to the originally sent CAN message, thus, to the topmost object in the sorted list.

If the data are identical, the transmission was successful. The topmost object is removed from the list and the controller proceeds back into the `idle` state.
If a CAN controller receives a message different from the one it originally transmitted, its message did not win media access, either due to other messages of higher priority or due to non-preemption of earlier started transmissions. In this case, another copy of the same topmost message on the `TreeMap` is retrieved. The variable `timeAccess` is set to the up-to-date system time and the controller enters the `idle` state, retrying transmission of this particular message as soon as the bus is idle again. This explains why only copies of the topmost object are retrieved from the list until transmission was successful. Otherwise, the highest priority data would be removed even though it might be needed for retransmission.

CAN Bus

The CSMA/BA mechanism and the transmission of messages is resembled in the `CAN Bus` active object. The bus is capable of receiving data from the ECUs, namely from the CAN controllers, and of transmitting messages as broadcast to all connected ECUs. This functionality is depicted by the bidirectional transition between `CAN ECU` objects and `CAN BUS` in figure 5.3.
In a real-life CAN bus system, each ECU is permanently listening to the communication channel and detecting by its own whether the bus is idle or busy. In the simulation model, the `CAN Bus` object is in charge of monitoring its state. The current value – idle or busy – is communicated to all connected ECUs, which are able to act accordingly in terms of sending to an idle bus or waiting when the bus is busy, complying with non-preemption in CAN data transfer. The unidirectional transition from `CAN Bus` to `CAN ECU` in figure 5.3 resembles this communication. The bus state is broadcasted via simple message objects of type `state_msg`, containing only one boolean variable `state`, which is set to "true" for an idle bus and to "false" in case the bus is busy.
Figure 5.8 shows the statechart of the `CAN Bus` active object. If no data is to be transmitted, the bus is in the `idle` state. As soon as a message arrives from any of the ECUs, the bus changes from `idle` state to `cs_phase` state for 6 bit times of carrier sensing. The message is put into a `TreeMap`, which is used for storing the messages, sorted by CAN-ID, with lowest ID of highest priority on top of the list. During the carrier-sense phase, the state of the bus remains `idle`, such that

other ECUs are able to transmit CAN messages to the bus, too. Along these 6 bit times, all incoming messages are stored in the `TreeMap`.

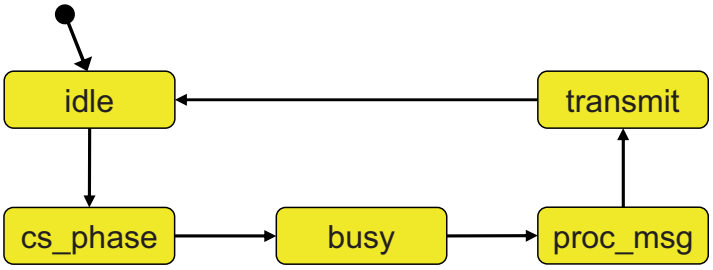


Figure 5.8: CAN Bus Statechart

Once the carrier-sense phase has elapsed, the bus changes from `cs_phase` state to `busy` state instantaneously. The respective `state_msg` object is broadcasted to all connected ECUs, which stop performing media access immediately. Entering the `proc_msg` state, all messages which have been stored in the sorted list during the carrier-sense phase are inspected. Obviously, the message with the lowest CAN-ID, thus with highest priority, is placed on top of the list. However, this message is not necessarily the one which was sent first during the 6 bit times of carrier sensing. To resemble real-life non-preemptive data transmission in the simulation model, the message with the oldest timestamp – with a time resolution of 1 bit time – from all messages in the `TreeMap` is the arbitration winner.
The system time of media access, as denoted in the variable `timeAccess` of a `can_msg` message object, of every single message in the sorted list is inspected. The message with the oldest timestamp – the smallest value of the double variable – is the winner of media access. In case two or more messages with an equal lowest timestamp are in the `TreeMap`, i.e. messages which were sent simultaneously from different ECUs, the one with the lowest CAN-ID, thus the highest priority, wins the arbitration. For inspection of the stored CAN messages no system time elapses, as the state transition from `busy` to `proc_msg` is implemented as a timeless, immediate transition.
On change to the `transmit` state, the respective message is removed from the sorted list and transmitted to all ECUs, which process the data as described beforehand. As CAN messages are assumed to be of maximum length of 130 bit, the timeout for the transition from `proc_msg` state to `transmit` state is chosen according to the message length and the actual CAN data rate. For example, for a 500 kbps CAN bus, the timeout computes to 0.00026 seconds. After transmission

of the arbitration-winning message, the `TreeMap` is cleared for the next carrier-sense phase and transmission cycle and the bus returns to the initial `idle` state. Figure 5.9 is intended to clarify the non-preemptive priority queuing and bus arbitration mechanism as implemented in the `CAN Bus` active object of the simulation model.

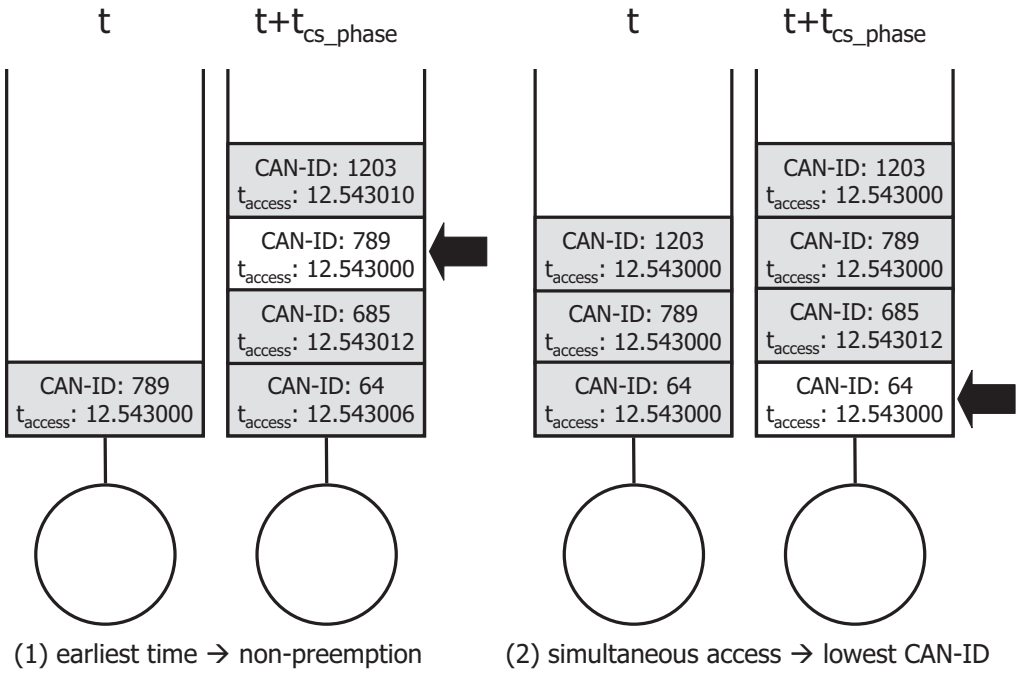


Figure 5.9: CSMA/BA Mechanism in the Simulation Model

Two scenarios are illustrated, namely non-preemption due to earliest start of transmission in (1) and a CAN-ID-based success in bus arbitration for simultaneous bus access in (2). The 6 bit time carrier-sense phase is assumed to be 0.000012 seconds for a 500 kbps CAN bus and the time annotations for bus access of the stored and sorted messages are denoted in seconds. Modeling of message-wise data transmission instead of processing bit-by-bit of a CAN message and the accuracy of AnyLogic™'s timestamp information are combined and allow for a reasonable and simulation-time-efficient resemblance of real-life CAN media access and data transmission.

5.4.3 FlexRay

On the top-level view, FlexRay communication is modeled in analogy to CAN, which means that several ECUs are connected to a serial bus for data exchange (cf. figure 5.10).

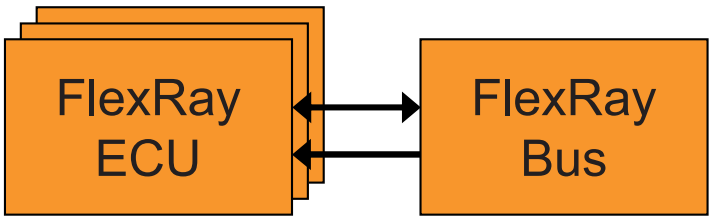


Figure 5.10: Conceptual Top-Level View on FlexRay Communication

Again, a stacked representation is used for the `FlexRay ECU` object to indicate that possibly more instances are connected to the same `FlexRay Bus` object. As described for the FlexRay system architecture in chapter 3, the TDMA-based data transmission scheme requires a global schedule, according to which media access and data transfer are organized for each of the ECUs. In the simulation model, the `FlexRay Bus` object is in charge of coordinating a global communication schedule by means of triggering ECUs for sending of data appropriately and of receiving and broadcasting messages.

Generation of FlexRay TDMA Schedule

For event-based CAN communication, each control unit itself is capable of appropriate behavior at media access in terms of carrier sensing, evaluation of bitwise bus arbitration and sending of data. For the TDMA-based data transmission at FlexRay, a more sophisticated organization of distributed communication is indispensable. From chapter 3, it is obvious that FlexRay data transfer is arranged in cycles, with each cycle containing a certain number of static and dynamic slots and again, each slot carrying FlexRay PDUs. To store this manifold nested communication information, the Java™ data type `LinkedList` from the class library `java.util` is particularly suited. Figure 5.11 illustrates the programmatic approach of storing the relevant schedule information by means of cascaded linked lists. As highlighted in the graphic, the focus of FlexRay operation in the simulation model is on PDU-wise data transmission in the static segment in the

TDMA schedule. The `LinkedList` objects containing the information on the schedule are variables of the `FlexRay Bus` object.

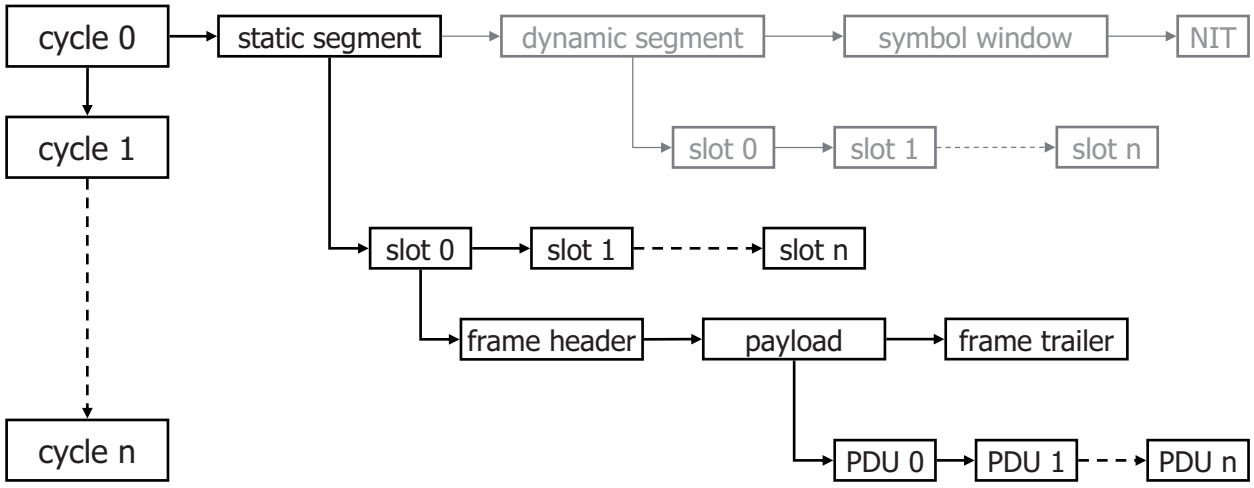


Figure 5.11: Organization of FlexRay Communication with Linked Lists

FlexRay ECU

Following the approach of modeling a CAN ECU, a FlexRay ECU is modeled by subdividing it internally into an active object representing the operating system (`ECU OS`) and a FlexRay communication controller (`FlexRay CTRL`). Figure 5.12 depicts the internal structure of a `FlexRay ECU` object.

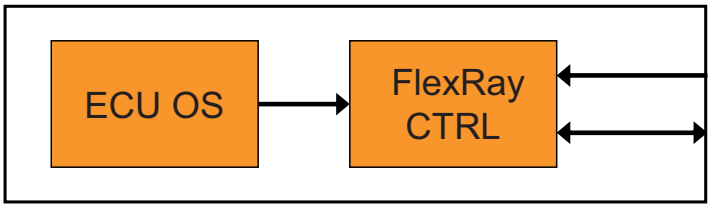


Figure 5.12: Structure of the FlexRay ECU Object

Operating System The same basic structure and behavior are employed for the operating system of a FlexRay ECU as already presented for the operating system of a CAN ECU in figure 5.5. The `ECU OS` object is capable of transferring the ECU from `sleep` to `awake` and of cyclic generation of communication data. An external file provides information on reasonable minimum and maximum

startup times for the respective ECU. The file is read in at start of a simulation run and a timeout value for ECU startup is sampled appropriately.
Of course, the `ECU OS` object of a FlexRay ECU differs from the `ECU OS` object of a CAN ECU as it generates message objects of type `fr_pdu`. Upon cyclic generation, the variables `name`, `sender`, `length` and `timeCreated` of the message objects are set accordingly. The `ECU OS` object contains a boolean variable `synchronized`, to accommodate the basic variations in automotive ECU operating system architectures:

- "true" resembles that tasks of the operating system can be synchronized to the time base of the FlexRay communication controller in the ECU and thus to the global time in the TDMA communication system, e.g. by employing a time-triggered AUTOSAR architecture [5]. This approach results in a rather short time gap between message generation in the operating system and transmission in the next TDMA time slot.
- "false" is the setting for event-based allocation of information, asynchronous to the communication schedule of the bus. This may result in a maximum waiting time for transmission of the current `fr_pdu` object of asymptotically one complete FlexRay cycle time, in case the operating system sends up-to-date data to the controller just when the current TDMA time slot for transmission passed by. This common issue of ECU design for TDMA-based communication can significantly effect the end-to-end transmission delays due to the large time gap between data acquisition and data transmission.

In general, this distinction is only necessary for a FlexRay ECU operating system and TDMA-based communication. For CAN, the CSMA/BA media access mechanism itself is designed such that it complies with event-based transmission requests. Hence, integrated synchronization from generation of CAN frames down to media access is not necessarily needed. In figure 5.12, the unidirectional transition from the `ECU OS` object to the `FlexRay CTRL` object resembles the flow of generated FlexRay PDUs from the operating system to the bus controller for data transmission.

FlexRay Controller The bus controller of a FlexRay ECU is modeled by the active object `FlexRay CTRL`. The controller is in charge of receiving messages

from the operating system of the ECU and transmitting these data to the FlexRay bus according to the global TDMA schedule. Of course, the controller is also responsible for reception and decoding of data from other communication partners. The functional behavior of this object type is depicted by the statechart in figure 5.13. From the `init` state, the controller changes into the `wait_for_slot` state after startup of the ECU. While a CAN controller tries to send data immediately upon arrival from the operating system, the FlexRay controller has to account for the schedule in a FlexRay communication system and thus has to wait until its dedicated time slot for data transmission begins. The schedule is organized in FlexRay communication cycles as denoted in figure 3.4, with each slot in the static segment being assigned to a particular ECU.

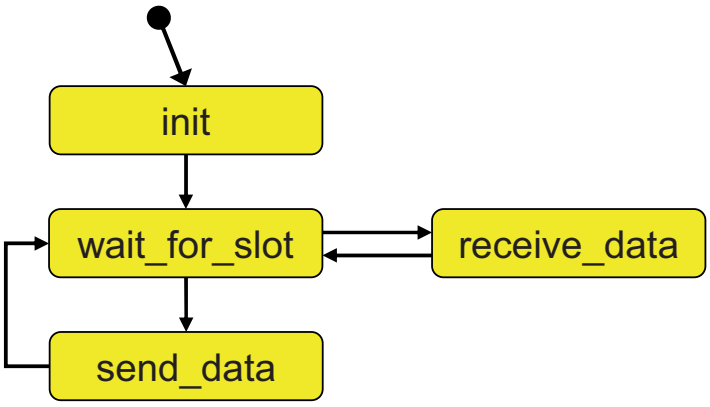


Figure 5.13: FlexRay Controller Statechart

When the dedicated time slot for data transmission begins, the internal state of the `FlexRay CTRL` object changes from `wait_for_slot` to `send_data` and the latest data available from the operating system is transmitted as `fr_pdu` object to the bus. When the PDU was sent, the controller returns to the `wait_for_slot` state again. At any time while the FlexRay controller is waiting for transmission capacity in the TDMA schedule, it is able to receive data from other FlexRay communication partners. This functionality is indicated by the bidirectional transitions from `wait_for_slot` to `receive_data`.
The functionality of a real-life FlexRay bus controller is implemented in the `FlexRay CTRL` object and the bidirectional transition from the `FlexRay CTRL` to the outside of the `FlexRay ECU` object in figure 5.12. As explained in the following, the additional unidirectional transition from the outside world to the `FlexRay CTRL` object in figure 5.12 is essential for information on central scheduling of TDMA-based data transmission.

FlexRay Bus

In reality, each FlexRay ECU is provided with the overall TDMA schedule and synchronized to a global time base in the communication system. Hence, a controller knows exactly at which time it is allowed to send data to the bus and what kind of data it can meanwhile expect to receive from other ECUs. In the simulation model, the assumptions are made that data transmission is performed in the static segment only and that the communication controller of a FlexRay ECU is perfectly synchronized to the common time base.
Various approaches are imaginable to implement data transmission of several ECUs via a time-triggered communication protocol in a simulation model:

- Each ECU is provided with the complete schedule and is permanently evaluating the actual system time, the progress in cycle- and slot-based communication and its own data generation and transmission. Of course, this yields a close resemblance of real-world FlexRay ECU operation. However, it might lead to a significant increase in computational load in the simulation, in case each controller has to evaluate its own state and the overall system state permanently.
- One central instance is responsible for coordination of communication. This incorporates working off one cycle after the other, evaluating which ECU is expected to send data in which dedicated slot and triggering of the respective ECUs appropriately in order to establish communication. Of course, these additional scheduling tasks must not consume any simulation time. Otherwise, the significant reduction of computational load comes to the price of inaccurate timing behavior of data transmission in the simulation.
- A mixture of centralized and decentralized coordination of scheduled communication, incorporating aspects of both other paradigms mentioned beforehand.

In the simulation model, the paradigm of one central entity for scheduling FlexRay data transfer was chosen. It appears to be the most promising approach in terms of computational performance of simulation runs, maintainability of the simulation model and ease of understanding the modeling of complex, time-triggered communication mechanisms. Consequently, the very core component of a FlexRay communication system is in charge of coordinating the TDMA schedule – the FlexRay

bus. According to the statechart as depicted in figure 5.14, the `FlexRay Bus` object works off one TDMA cycle after the other, virtually triggering the FlexRay ECUs to send data in the dedicated time slots. The latter task is achieved by broadcasting `fr_config_data` messages, which are basically of type `fr_pdu`. The sender is set to `FLEXRAY` and `name` to the name of the PDU which is to be sent next in the schedule. The ECUs evaluate the data and the one which is actually addressed by the `fr_config_data` message transmits its current data to the bus. No system time elapses for this as the central scheduling messages of type `fr_config_data` are not part of the actual FlexRay data transmission. No real transmission time is consumed by these messages, thus they do not contribute to the overall FlexRay bus utilization.
The statechart for FlexRay operation as illustrated in figure 5.14 represents the complete TDMA data transmission with the additional functionality of central scheduling. On change from `init` to `awake`, FlexRay bus operation is started, which means the `FlexRay Bus` object initializes data structures for TDMA operation, e.g. lists of cycles, slots and scheduled ECUs.
A TDMA communication cycle starts upon entering `start_cycle`, beginning to process the static slots in `start_slots`. First, the header of a FlexRay frame in the slot is sent in `send_header`, then the payload is inspected in `send_payload`. In `get_data`, the next datum within the FlexRay frame is evaluated. For payload (state `payload`), the sending ECU is determined and forced to transmit its current data to the bus by triggering the respective ECU via a `fr_config_data` message. If the desired data from this ECU is received, the bus proceeds to `data_ready` state and broadcasts the current PDU via the state `broadcast` to all ECUs in the system. A timed transition from `broadcast` to `send_payload` resembles the transmission delay according to the actual payload size and the data rate.
If either the whole FlexRay frame or a remaining part of the frame is idle (1) – i.e. the slot is not assigned to any ECU or was not filled with payload completely – or the complete payload with no remaining idle space was processed (2), the bus enters the `send_trailer` state via `idle_data` for (1) or `send_payload` for (2) and broadcasts the trailer of the current FlexRay frame. From `send_trailer`, the bus advances to the next TDMA time slot. If all slots of the current TDMA cycle have been worked off, the transition to `slots_finished` is activated and the system proceeds to the next TDMA cycle.

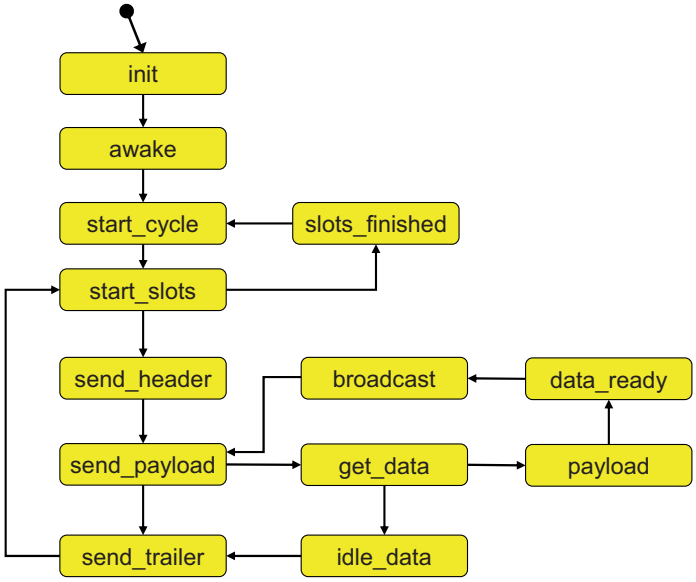


Figure 5.14: FlexRay Bus Statechart

5.4.4 Gateway

The model of the gateway incorporates the functionality as described in chapter 3 and the structure as depicted in figure 3.6. The top-level view on the `Gateway` active object in figure 5.15 shows the internal composition, consisting of a `Gateway Switching Unit` object and one or more `Gateway Bus CTRL` objects.

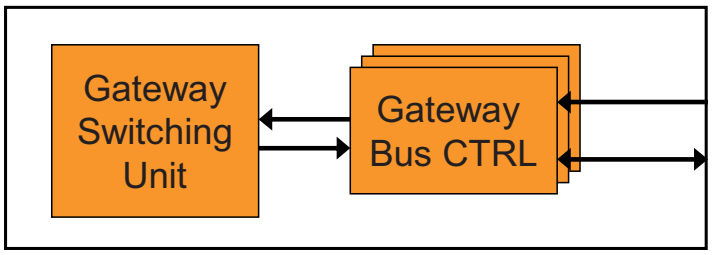


Figure 5.15: Conceptual Top-Level View on Central Gateway

The gateway can be connected to several bus systems, which is indicated by the stacked representation of the bus controller. Each of the `Gateway Bus CTRL` objects can be of type `CAN CTRL` or `FlexRay CTRL`, implementing exactly the same functionality as the controller in a `CAN ECU` or `FlexRay ECU` object. Instead of connection to an ECU operating system, each single bus controller of the gateway is linked with the `Gateway Switching Unit` object.

The switching unit implements the routing mechanism in the gateway, i.e. how incoming data from various connected buses is stored, processed and forwarded. As for automotive gateways no commonly standardized architecture is available, a choice of the two most prevalently employed technological strategies is implemented in the simulation model. Both internal data processing strategies by cyclic polling of connected bus ports and by interrupt- and priority-based routing from one central queue are explained in the following.

Cyclic Polling Routing Strategy

A cyclic polling routing strategy in the gateway implements a periodical scanning of ports of connected buses for data to be routed. The statechart in figure 5.16 depicts the behavior of a gateway switching unit with a cyclic polling routing strategy.

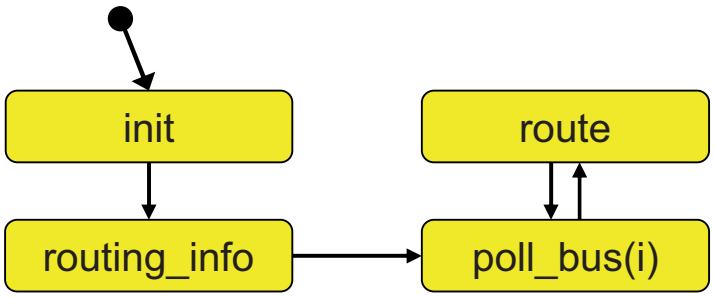


Figure 5.16: Switching Unit Statechart for Cyclic Polling

Along the transition from `init` to `routing_info`, the gateway processes routing information and stores it in an appropriate data structure, e.g. a look-up table. This allows for determining for any message that arrives at the gateway, whether it has to be routed to any other buses than the bus it originates from. Subsequently, the gateway starts operation by polling ports of connected buses cyclically for any newly arrived communication data in state `poll_bus(i)`. If applicable, data is routed to destination buses in state `route`. The process of scanning one bus port after the other is indicated by the variable i in state `poll_bus(i)`. For n buses connected to the gateway, each bus port is assigned a number from 0 to $n-1$. Upon entering state `poll_bus(i)`, i is set to zero and, while $i \leq n-1$ holds, increased by one each time the transition from `route` to `poll_bus(i)` is activated, or reseted to zero if $i = n-1$ applies.

Priority-Based Routing Strategy

If messages are to be routed according to a certain ordering in priority, a necessary prerequisite is that all single messages or message types in the network share a common attribute, which allows for deriving a ranking of data from it. In a communication system composed of CAN and FlexRay buses, the CAN-ID can be such a characteristic for two reasons:

- A CAN frame, received by the gateway from any of the CAN buses, already contains a CAN-ID. As these identifiers are assigned network-wide uniquely, an ordering of CAN messages can easily be derived from already existing CAN-IDs.
- A FlexRay PDU, which arrives at the gateway and is to be routed to one or more CAN buses, will essentially obtain a CAN-ID for transmission to the sink CAN bus(es). Again, this CAN-ID allows for an unambiguous sorting of the message into a single priority queue.

If messages are to be routed in a scenario where a CAN bus is neither source nor sink of the data, e.g. between two separate FlexRay buses, a more sophisticated priority-based routing strategy must be employed in the gateway[1]. A priority-based routing mechanism is basically event- or interrupt-based, which means that processor performance of the gateway is only needed if data is actually to be routed. This is oppositional to the paradigm based on cyclic polling, where bus ports must be scanned regularly, regardless of effective data inside.
The statechart in figure 5.17 depicts the functional segmentation of a purely priority-based routing. If the transition from `init` state to `routing_info` is executed, routing information is processed and stored appropriately by setting up a routing table. The first message arriving at the gateway triggers the immediate transition from `routing_info` to `data_available`. The message and all on-following ones are stored in the switching unit using a hashing-based approach. Therefore, a `TreeMap` from the Java™ class library `java.util` is employed and messages are ordered according to the CAN-ID for either the source or the sink bus.

[1] For example, all messages with an accessible CAN-ID may be sorted into the routing list according to this ID, all other data might be assigned "dummy-IDs" for routing, which are chosen either out of the set of network-wide idle CAN-IDs or outside of the range of 11-bit standard CAN-IDs.

If the `TreeMap` is non-empty, the topmost message is removed along the transition from state `data_available` to `get_top_prio` and routed according to the information from the routing table when entering state `route`. A dedicated timeout for proceeding back from state `route` to state `data_available` resembles the whole process of removing data from the queue, retrieving routing information and copying data to the controller objects of destination buses. Having returned to state `data_available`, the now topmost message is removed and the routing process is started over.

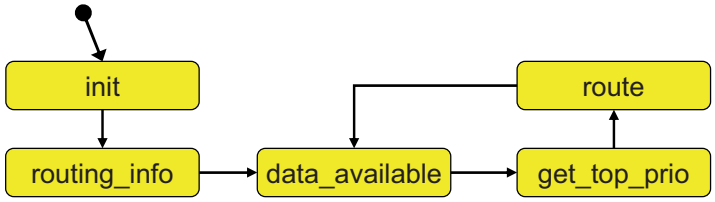


Figure 5.17: Switching Unit Statechart for Priority-based Routing

5.4.5 Overall In-Car Communication Network

The core components of the simulative investigations on in-car data transmission have been presented in this chapter, namely message objects representing actual communication data (`can_msg` and `fr_pdu`) and active objects resembling the respective parts of the network infrastructure (`CAN ECU` and `FlexRay ECU`, both with operating system and bus controller, `CAN Bus`, `FlexRay Bus` and a `Gateway`).
Consequently, a simulation model of an automotive communication system must be assembled of these components. Figure 5.18 depicts a possible structure of a network model with one central gateway, which is represented as an active object according to the top-level view in figure 5.15. The stacked representation of bus controllers in the gateway, of buses and of ECUs is again used to emphasize that a multitude of these devices can be joined towards an all-encompassing model.

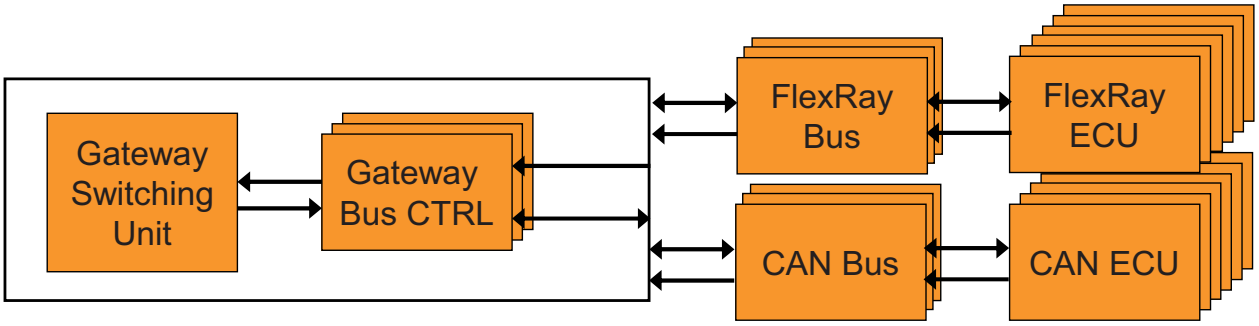


Figure 5.18: Conceptual Top-Level View on In-Car Communication System

5.5 Discussion of Discrete Event Simulation Approach

A simulation model can never be a one-to-one copy of the real-world system. At various levels of system modeling, structure and behavior have to be abstracted and simplified in order to make the simulation approach applicable, to keep the modeling efforts worthwhile and to avoid excessive runtimes of simulations due to an "over-engineered" granularity in modeling.
Automotive communication systems reveal some characteristics, for which discrete event simulation appears to be particularly suited. First and foremost, the cyclic broadcasting of data at CAN and FlexRay allows for a processing of simulation events and internal state updates according to a strictly ordered, discretely timed event list. Yet, additional or unforeseen events, like failures or external stimuli, could easily be included in the event list in addition to the fix planned, periodic tasks. The simulation tool AnyLogic™ was a very good choice for modeling for several reasons:

- AnyLogic™ supports an integrated modeling of the system, using the discrete event simulation paradigm. This allows for resembling real-world ECUs, CAN, FlexRay and a gateway on a level of abstraction fulfilling the demands of reasonable detailedness and sufficient simulation performance.
- The relevant functional and structural entities of the system can easily and clearly be modeled using a statechart-based approach. The segmentation of a physical device into its hardware and, from a functional point of view, its most characteristic behavioral patterns, yields a structured, adaptable and understandable model.

- The transitions between single states can be annotated with (almost) arbitrarily chosen timing constraints. Several aspects in an in-car communication system have shown to be time-dependent, e.g. controller startup durations, cyclic generation and transmission of data and routing between bus systems. With the ability to annotate deterministic timeouts for state transitions or to sample values from stochastic distributions, the system operation can be well reproduced as it evolves over time.
- The integration of Java™ and the full compatibility with programming constructs and libraries allows for employing a widespread, advanced programming language. The object-oriented programming supports the modeling significantly, e.g. by simple instantiation of tens of messages or by using constructs for data storage and maintenance in the simulation model.
- In addition, AnyLogic™ offers capabilities for a sophisticated simulation control. Whenever a system with partly stochastic characteristics is to be investigated, a reasonable number of replications of a simulation run with varying *random seeds* is indispensable in order to obtain trustworthy outcomes. In AnyLogic™, several constructs are available to automate replications by permanent evaluation of the level of confidence of achieved performance measures.

The single components in the simulation model are designed modularly and feature standardized interfaces, which significantly contributes to flexibility in composition of a communication system out of the single devices. The structure of the model is easy to be adapted, which is one merit of the intuitive modeling environment in AnyLogic™. With respect to the capabilities and the performance of a statechart-based, discrete event simulation approach, the communication networks investigated in this thesis are only an exemplary sample for technical relevant, real-world-related in-car network topologies.

6 Worst-Case Analysis of In-Car Data Transmission

Research on worst-case analytics for in-car data transmission was carried out together with Dr. Kai-Steffen Hielscher, Dr. Ulrich Klehmet and Prof. Dr. Reinhard German from *Chair of Computer Science 7, University of Erlangen-Nürnberg*. Results were published in [27], [28], [29] and [21].

6.1 Motivation

The approaches that have been presented and discussed in the previous chapters of this thesis, namely real-world measurement studies and discrete event simulation, are well-suited to capture the dynamic behavior of networked, modern in-car electronics, especially regarding data transmission for distributed applications. The outcomes from these investigations are closely related to daily operation of the system, which was already introduced beforehand as the typical *use-case*. For many automotive applications which are not safety-critical, like comfort- or entertainment-oriented functions, it is sufficient to know the characteristics of in-car data transmission in these normal operation scenarios.
For example, occupants will hardly realize a deviation of a few tens of milliseconds in delay between manipulation of an entertainment device and actual system reaction. Moreover, variations in communication delays for safety-uncritical functions are not likely to expose passengers to serious threats in terms of system malfunctions or instable and dangerous driving situations. Quantile-related information on system behavior, as it can be derived from measurements and simulation studies, has shown to be a reasonable part in engineering of such applications, e.g. "in 99% of activation, an entertainment device will react after a maximum of 100 milliseconds to a customer's input".

For time- and safety-critical applications from the fields of vehicle safety, a more all-encompassing view on system operation is indispensable. Of course, quantiles for performance measures are a worthwhile component in system design and evaluation of expectable system behavior. Yet, depending on the criticality of the actual intelligent safety application – especially in terms of unmotivated or significantly delayed activation – the absolute worst-case operation scenario must be incorporated. Timing aspects of these networked precrash functions are a key issue for application performance and in turn for the efficiency of future vehicle safety concepts. Consequently, a worst-case investigation must be related to the worst-case behavior of data transmission in the distributed system of ECUs.
The pitfall of both simulation and measurement studies is that, regardless of the number of replications or recorded data samples, one can never assure that the computed maximum values for performance measures, like transmission delays, buffer sizes or packet losses, were achieved having covered the real worst-case constellation of possible system behavior. Hence, other approaches must be employed to round off the quantile- and measurement-based system evaluation, with respect to determination of reliable thresholds for system performance in *all* operation scenarios.
Several analytical methods are available, which allow for computation of guaranteed worst-case results for performance measures. Typically, purely analytical and deterministic algebraic constructs are applied to a set of system-dependent input parameters, like transmission capacity, communication schedule or service strategy. No stochastic aspects play into the analyses, such that the outcomes are not prone to a remaining "statistical uncertainty" and yield valid upper (or lower) bounds on system performance. In this thesis, the analytical method of *Network Calculus* is adapted to the demands of typical automotive communication networks and applied for worst-case performance evaluation of time-critical in-car data transmission.

6.2 Network Calculus

The method of Network Calculus offers a theoretical framework for deterministic performance evaluation of computer networks. It represents a theory for investigations on deterministic queuing systems and is applied to these systems comparable to classical system theory from electrical engineering for analyses of

electronic circuits. In Network Calculus, the operation of *addition* known from system theory is replaced by the *computation of a minimum* and the *multiplication* from system theory becomes the *addition*.
As depicted in figure 6.1, Network Calculus is typically applied to evaluate a system with an input $x(t)$ and an output $y(t)$. Both $x(t)$ and $y(t)$ are regarded as cumulative flows, which means that the number of bits, packets, jobs, etc. arriving to or leaving from the system is increasing along time of observation. This increase is typically denoted by appropriate functions, for example a step function. Except for the service strategy, the internal characteristics of the system may be widely unknown.

Figure 6.1: System-theoretical View on Network Calculus

The method can be seen as a *worst-case analysis* for determination of performance bounds for a system in a worst-case operation scenario. Emanating from computer network engineering, typical performance measures to determine bounds for are:

- The (end-to-end) transmission delay. Applying Network Calculus to a communication system allows for determining the guaranteed maximum transfer time a message, packet, etc. will experience from sending to reception.

- The traffic backlog in one device or in the overall system. Knowing the maximum possible number of data which is in transfer in the system, supports adequate dimensioning of buffers or other storage components and in turn to avoid losses due to overflows.

- The minimum system output. For many systems and applications, it is essential that the output of data at the edge of a system never under-runs a certain lower limit, e.g. for constant further processing. By means of worst-case analyses, these guaranteed lower thresholds can be determined.

These performance-bounding values – characterizing worst-case behavior of traffic flows – are valuable information, whenever systems are to be evaluated for which, due to the criticality of decreased performance, all possible scenarios of operation must be considered. The remaining part of this chapter is intended to give an overview on the most important constructs of Network Calculus, especially on those, which were employed to analyze time-critical in-car data transmission. A comprehensive overview on Network Calculus can be found in [38], [9] or [10].

6.2.1 Theoretical Foundations

Network Calculus is an analytical method and algebraic constructs are an elementary part of this deterministic performance evaluation approach. The most important modeling elements in Network Calculus are the *arrival curve* and the *service curve*. In addition, the operation of *min-plus convolution* is of central interest.
For all following definitions and explanations, it is assumed that there is a flow F of bits, messages, etc. into a system S. $x(t)$ shall be the amount of data of F arriving in the time interval $[0,t]$ and $y(t)$ the amount of data leaving S in the time interval $[0,t]$. $x(t)$ is the arrival function of flow F. By definition, $x(0) = 0$ and $x(t) \geq x(s)$ for all $t \geq s$.

Arrival Curve

In most cases, the performance of a system is strongly influenced by the load, which is placed on the single components or on the overall system. Thus, the maximum number of arriving jobs, messages, etc. is an important modeling parameter in Network Calculus. To estimate which is "the maximum number", a specific function $\alpha(t)$ – the so-called *arrival curve* – is determined, which resembles an upper bound to the input $x(t)$ and fulfills the following:

Let $\alpha(t)$ be a non-negative, non-decreasing function. A flow F is constrained by or has the arrival curve $\alpha(t)$ if and only if

$$x(t) - x(s) \leq \alpha(t-s) \qquad \forall t \geq s \geq 0$$

In other words, the arrival curve $\alpha(t)$ must be chosen such that the cumulative input flow $x(t)$ is kept below this function at any time, i.e. it is guaranteed that at

no time *more* data arrives to the system than described by $\alpha(t)$. To achieve this, it is necessary to describe the arrivals of flow $x(t)$ mathematically with a closed formula for a valid arrival curve $\alpha(t)$.

An example for a commonly used arrival curve in Network Calculus is the so-called *token bucket function* according to

$$\alpha_{r,b}(t) = b + rt \qquad \text{for } t > 0 \text{ and } 0 \text{ otherwise}$$

In a token bucket function, a bucket is assumed as an abstract container, implementing a control mechanism that dictates when traffic can be transmitted. The bucket is of size b, which means it can contain at most b tokens. Each of the tokens represents a unit of bytes or a single packet of predetermined size. A specified amount of tokens is removed for the ability to send a packet and the bucket is refilled with tokens according to a constant rate r. If there are no tokens in the bucket, a flow cannot transmit its packets. The peak burst rate, up to which a flow can transmit traffic, is limited by the bucket size b. The token bucket function is often confused with the *leaky bucket function*, both traffic shaping mechanisms are adequately described in [39]. Figure 6.2 depicts an arrival curve $\alpha(t)$ of token bucket type, which forms an upper limit for input $x(t)$ with rate r and instantaneous burst b immediately after start time $t = 0$.

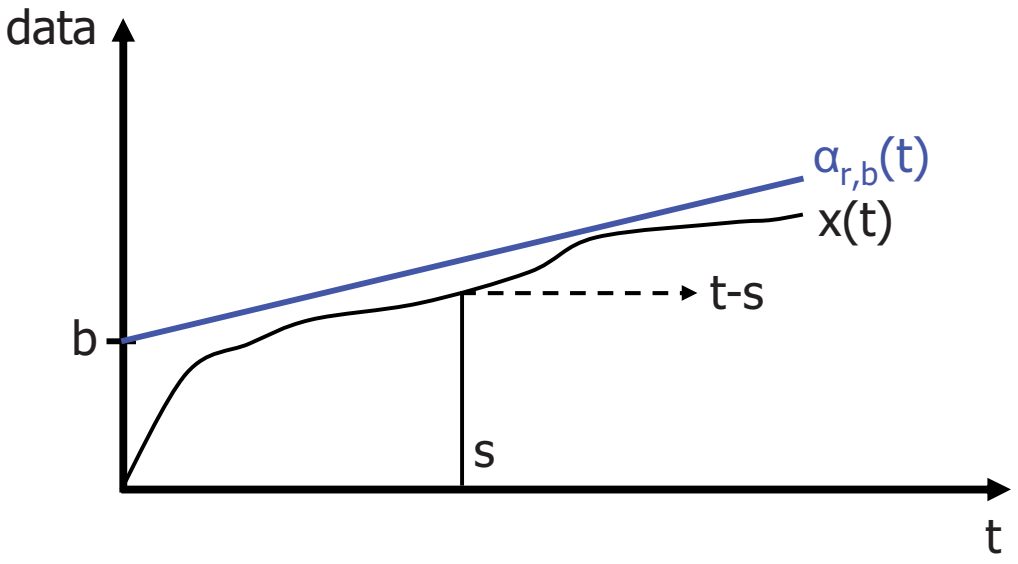


Figure 6.2: Token Bucket Arrival Curve

Applying the general definition of an arrival curve to a modeling scenario with a token bucket constraint yields:

$$x(t) - x(s) \leq \alpha_{r,b}(t-s) = b + r(t-s)$$

For $\Delta t := t - s$ and $\Delta t \to 0$ it holds that

$$\lim_{t \to s}\{x(t) - x(s)\} \leq \lim_{\Delta t \to 0}\{r \cdot \Delta t + b\} = b$$

Min-Plus Convolution

An important operation in Network Calculus is the *min-plus convolution*. *Min* hints at the computation of the minimum, *plus* stands for addition, which replaces the operation of multiplication in convolution of classical system theory.

The min-plus convolution of two non-negative, non-decreasing functions $f(t)$ and $g(t)$, which fulfill $f(t) = 0$ and $g(t) = 0$ for $t \leq 0$, is defined as:

$$(f \otimes g)(t) = \inf_{0 \leq s \leq t}\{f(s) + g(t-s)\}$$

The min-plus convolution can be applied to the arrival curve $\alpha(t)$ and the cumulative input flow $x(t)$, in order to emphasize that $\alpha(t)$ is a valid upper bound for $x(t)$ at any time t:

$$x(t) \leq (x \otimes \alpha)(t)$$

Service Curve

The concept of arrival curves describes an upper bound for an input stream to a system processing some type of data. Thus, the input can be bounded and handled mathematically. However, the performance of a system is also strongly influenced by the service capabilities, i.e. the speed and the strategy in processing of incoming jobs. The function $\beta(t)$ – the *service curve* – is the modeling element which describes the service offered by a system. As shown later, the service curve is essential for determining the worst-case performance of the system, e.g. in terms of estimating the guaranteed minimum output $y(t)$ or determining worst-case backlog and delays. The definition of a service curve, which is compliant to the theory of Network Calculus, is as follows:

Given is a system S with input flow $x(t)$ and output flow $y(t)$. The system offers a (minimum) service curve $\beta(t)$ to the flow if and only if $\beta(t)$ is a non-negative, non-decreasing function with $\beta(0) = 0$ and $y(t)$ is lower bounded by the min-plus convolution of $x(t)$ and $\beta(t)$:

$$y(t) \geq (x \otimes \beta)(t)$$

Obviously, the service curve $\beta(t)$ must be chosen such that processing the input $x(t)$ according to $\beta(t)$ – which is expressed by convolving $x(t)$ and $\beta(t)$ – yields the minimum expectable system performance, hence a lower bound for the system output $y(t)$.

One commonly used service curve is the *rate-latency function*, defined as:

$$\beta(t) = \beta_{R,T}(t) = R \cdot [t - T]^{+} := R \cdot \max\{0; t - T\}$$

The rate-latency function reflects a service element, which offers a minimum service of constant rate R after a worst-case latency of T. In figure 6.3, the green-colored graph $\beta_{R,T}(t)$ is a rate-latency service curve with rate R and worst-case latency T.

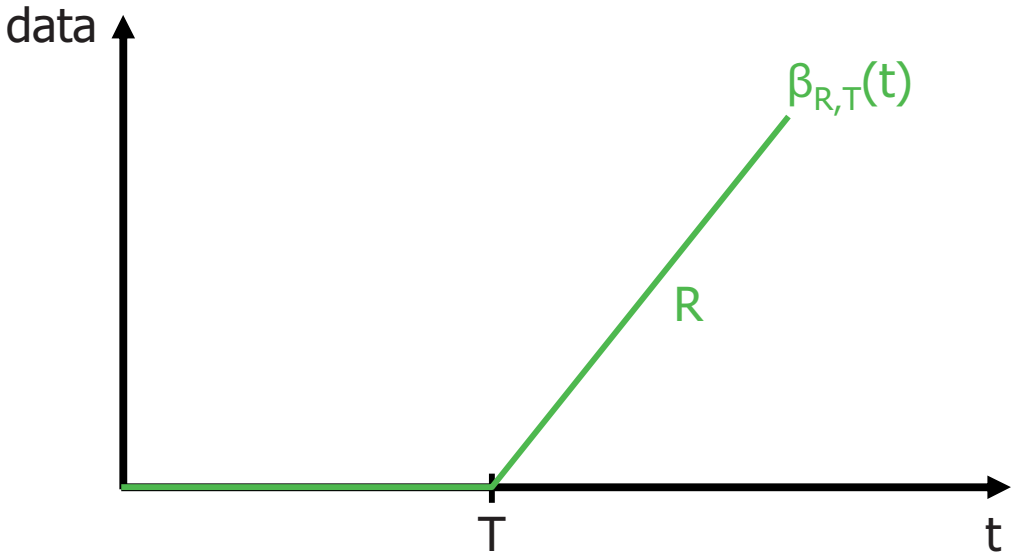


Figure 6.3: Rate-Latency Service Curve

The arrival curve and the service curve are the core modeling elements for a worst-case performance evaluation using Network Calculus. Of course, the guaranteed

worst-case performance bounds for a system can only be as good as the functions $\alpha(t)$ and $\beta(t)$ resemble the traffic and service characteristics. Hence, considerable effort must be spent in order to derive mathematical system descriptions that fit closely to the actual system behavior.

Bounds on Performance Measures

With the modeling elements arrival curve and service curve, together with the min-plus convolution, it is possible to derive bounds for performance measures for a system in a worst-case operation scenario. Especially for systems which are expected to fulfill any real-time demands, knowing the performance capabilities in terms of maximum transmission times and/or maximum buffer sizes is valuable information at system design. Applying Network Calculus, upper bounds can be derived for the transmission delay and for the backlog of jobs in process in the system.

Delay Bound Assume a flow $x(t)$, constrained by an arrival curve $\alpha(t)$, passes a system which offers a service according to the service curve $\beta(t)$. The maximum delay d is given as the supremum of all possible delays of data, i.e. is defined as the supremum of the horizontal deviation between the arrival curve and the service curve:

$$d \leq \sup_{s \geq 0} \{\inf\{\tau : \alpha(s) \leq \beta(s+\tau)\}\}$$

To illustrate this, suppose there is a system with input $x(t)$ that is upper-bounded by a token bucket arrival curve $\alpha_{r,b}(t)$, thus $x(t) - x(s) \leq \alpha_{r,b}(t-s)$, and a service that is resembled by a rate-latency function $\beta_{R,T}(t)$:

$$y(t) \geq \inf_{s \leq t} \{x(s) + \beta_{R,T}(t-s)\}, \quad r \leq R$$

Based on the theorems introduced beforehand, the delay bound d can be determined as $d \leq b/R + T$. Figure 6.4 depicts both the token bucket arrival curve $\alpha_{r,b}(t)$ and the rate-latency service curve $\beta_{R,T}(t)$. The worst-case delay d is denoted as horizontal, dashed, double-headed arrow, remarking the maximum horizontal distance between arrival curve and service curve.

Backlog Bound The backlog is the amount of data that is held inside the system. If the system is a single buffer, it is the queue length, if the system is more complex, then the backlog is the number of data in transit, assuming that input and output can be observed simultaneously.
Considering a system S with input flow $x(t)$ and output flow $y(t)$, where $x(t)$ is constrained by an arrival curve $\alpha(t)$ and S offers a service according to a service curve $\beta(t)$. The backlog v at time t, $v(t) = x(t) - y(t)$, is bounded by the supremum of the vertical deviation of arrival curve $\alpha(t)$ and service curve $\beta(t)$:

$$x(t) - y(t) \leq \sup_{s \geq 0} \{\alpha(s) - \beta(s)\}$$

For a system with token-bucket-constrained input and rate-latency service, the backlog v is actually bounded by $v = b + rT$. In figure 6.4, the vertical, dashed, double-headed arrow resembles the maximum deviation between arrival curve and service curve – the worst-case backlog v of jobs, etc. in the system.

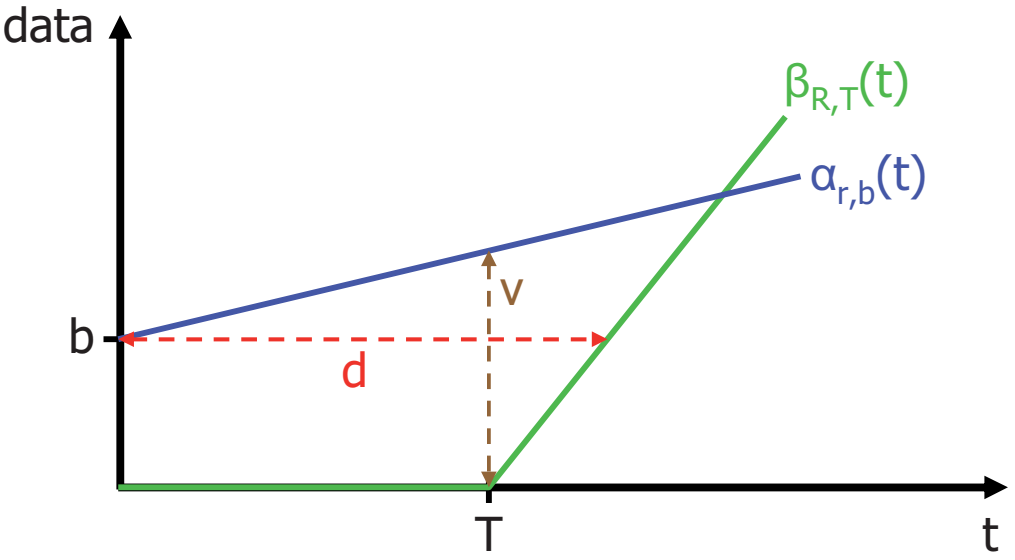


Figure 6.4: Delay Bound and Backlog Bound

Strict Service Curve For many classes of systems, networks and network nodes where backlog can appear, it is important that the service curve fulfills the conditions of a *strict service curve*, which is, according to [38], defined as:

A system S is said to offer a strict service curve $\beta(t)$ to a flow $x(t)$ if, during any backlogged period $[t_s, t_e]$ with start time t_s, end time t_e and $t_e - t_s = u$, it holds

that $y(t_e) - y(t_s) \geq \beta(u)$, i.e. the output flow is at least equal to the minimum service.

Concatenation If a flow traverses a system with two network elements, each one offering a specific service according to β_i, $i = 1,2$, the *concatenation* of the two systems offers a service curve of $\beta_1 \otimes \beta_2$ to the flow.

For example, imagine two nodes with rate-latency service curves β_{R_i,T_i}, $i = 1,2$. The concatenation of these nodes yields:

$$\beta_{R_1,T_1} \otimes \beta_{R_2,T_2} = \beta_{min(R_1,R_2),T_1+T_2}$$

The concatenation theorem allows to understand a phenomenon known as *Pay Bursts Only Once*, which states that the bounds obtained by considering the concatenated service curve are tighter than the bounds obtained by considering every network element in isolation and addition of the individual bounds.

6.3 Application of Network Calculus to CAN Communication

As motivated in chapter 3, the priority-based CSMA/BA media access at CAN is a potential source of delay. The real-life measurements presented in chapter 4 substantiated this effect in typical use-case scenarios. In order to compute valid upper bounds for CAN transmission delays in worst-case operation scenarios, Network Calculus was applied to CAN communication systems.

6.3.1 Input Data

CAN ECUs typically communicate cyclically, i.e. CAN messages are broadcasted to the bus in pre-defined fix time intervals[1]. For each message, two important CAN parameters are accessible, which are sufficient to apply Network Calculus, namely the CAN identifier (CAN-ID) and the cycle time.

[1] Some messages may also be sent *on-event*, which depends on the actual driving situation and is not considered in the traffic analysis in the following.

One remark regarding the CAN-ID: In the following, the distinction between CAN-ID and *priority class* (or simply *priority*) is made. The CAN-ID is the actual identifier of a certain message, ranging from 0 to 2047 for standard 11-bit CAN-IDs. In a set of CAN messages, each single message has its own, unique CAN-ID. A ranking according to the priority can be established by ordering the messages in the data set from lowest to highest ID in use. Priority class 0 corresponds to the lowest CAN-ID and hence has highest priority, class 1 complies with the next-lowest ID, class 2 with the third-lowest ID and so on.

6.3.2 Generation of Arrival Curves

A central issue of performance evaluation with Network Calculus is the generation of appropriate arrival curves for data traffic. For cyclically sent messages, CAN media access will also be performed in a cyclic manner. Ideally, the bus is idle when an ECU attempts transmission and the respective data can be sent right away. If two or more ECUs are trying to sent CAN messages simultaneously, the individual priorities determine, which message will be sent first, second, third, etc. Depending on the number of messages accessing the bus simultaneously, this might cause considerable delays in data transmission for messages which lose CSMA/BA media access several times consecutively.
To determine the worst-case delays at media access for all cyclic messages on a CAN bus, the priority-wise cyclic data transmission has to be captured with adequate arrival curves.

Therefore, let P denote the number of the lowest priority and $p \in \{0,1,\ldots,P\}$ be a priority for one particular CAN message. Basically, three types of priorities must be distinguished: The *own* priority p (a certain CAN message belongs to), the *higher* priorities from 0 to $p-1$ (winning the competitive media access if sent simultaneously to priority class p) and the *lower* priorities from $p+1$ to P. The lower priority classes are dominated by class p at media access and can be neglected in the following.
Simultaneous arrivals of higher priority data from classes 0 to $p-1$ affect the transmission of messages from class p and need to be considered when computing worst-case delays for a message of class p. An individual, cumulative arrival curve must be formed for each priority class higher than p. The superposition of all $p-1$ arrival curves of higher priority classes yields the overall amount of

data sent in a specified time interval, during which no media access is possible for class p due to higher prioritized traffic[2]. The generation of arrival curves is based on the assumption that at each discrete point of time, all messages with matching cycle period occur at once.
For instance, assume four ECUs ECU1 to ECU4, sending the messages MSG1 to MSG4 in cycles of 5 ms, 10 ms, 15 ms and 20 ms. Table 6.1 depicts the actual scheduling, i.e. at which time instance which messages are assumed to be sent simultaneously in this exemplary scenario, which is indicated by "x" as an entry in the respective column.

Table 6.1: Example for Simultaneous CAN Media Access

	Message (Cycle Time (ms))			
Time (ms)	MSG1 (5)	MSG2 (10)	MSG3 (15)	MSG4 (20)
0	x	x	x	x
5	x	-	-	-
10	x	x	-	-
15	x	-	x	-
20	x	x	-	x
25	x	-	-	-
30	x	x	x	-
⋮	⋮	⋮	⋮	⋮

Except for non-preemption of transmissions in progress, simultaneous sending is the explicit event causing delays at media access. As performance evaluation with Network Calculus is specifically aiming at determination of upper, worst-case bounds on performance measures, the underlying constellations of system operation must be identified. A valid constellation is that *all* ECUs are sending data simultaneously, yielding the maximum number of collisions at media access and in turn the maximum delays. Of course, this assumption requires that all ECUs start sending at once at a global system time $t = 0$.

Continuing in the formal construction of arrival curves, it holds that a CAN message of priority p is sent in cycles of length c_p. Without loss of generality, c_p are

[2]The highest priority class 0 has to be treated separately, since here the arrival curve for higher classes is constantly zero as no higher priorities exist.

integer values in the following, depending on application specific demands for update rates of information exchange between ECUs. A new message of priority p will be sent immediately after times $k \cdot c_p, k \in \mathbb{N}_0$, i.e. at integer multiples of c_p. Let C denote the *least common multiple (lcm)* of all c_p. At this point of time, messages of all priorities are sent, and since all messages are sent in a cyclic manner, the cycle of message arrivals will repeat in the same sequence for all messages of all priorities after the total cycle time $C = \text{lcm}\{c_0, c_1, \ldots c_P\}$.

A step function is particularly suited to resemble the cyclic sending of data and the increasing amount of data sent cumulatively along time. The arrival curve $\alpha_p(t)$ for a message of priority class p can be written as the following step function:

$$\alpha_p(t) = \left\lceil \frac{t}{c_p} \right\rceil \cdot l$$

Note that l denotes the maximum length of a CAN message, which is 136 bits, including the maximum number of stuff-bits and the 6-bit carrier-sense phase.
The cumulative arrival curve $\hat{\alpha}_p(t)$ resembles *all* arrivals of data with priority higher than p. Consequently, it is constructed as the superposition of all individual arrival curves:

$$\hat{\alpha}_p(t) = \sum_{i=0}^{p-1} \alpha_i(t) = \sum_{i=0}^{p-1} \left\lceil \frac{t}{c_i} \right\rceil \cdot l$$

6.3.3 Determination of the Service Curve

Having determined a step function as an arrival curve, which bounds the input flow appropriately, an adequate service curve must be found in order to reflect the behavior of CAN data transmission. Like in many other worst-case modeling scenarios, a rate-latency service curve $\beta_{R,T}(t)$ is an adequate choice.
First and foremost, a CAN bus offers a fix data rate R to all CAN messages, e.g. for a low-speed CAN 125 kbps and for a high-speed CAN 500 kbps or 1 Mbps. Thus, the parameter R of the rate-latency service curve can easily be determined knowing the data rate of the CAN bus.
As data transmission in a CAN bus system is non-preemptive, the worst-case latency T in $\beta_{R,T}(t)$ is determined as follows:

Assuming a worst-case scenario, a CAN message *always* has to wait until a transmission in progress of another CAN message – possibly of lower priority – is completely finished, i.e. it is always subject to non-preemption and finds the bus never idle when attempting data transmission. The maximum non-preemption period can be computed from the given CAN data rate and the maximum frame size of 136 bits. For example, for a high-speed CAN with a data rate of 500 kbps, T is determined as:

$$T = l/R = 136 \text{ bits}/500 \text{ kbps} = 0.000272 \text{ s}$$

The CAN data rate in this example is set to 500 kbps, since all following investigations focus on high-speed CAN bus systems as typically employed in the automotive industry. For the rate-latency service curve $\beta_{R,T}(t)$ of a CAN communication system, it holds:

$$\begin{aligned} \beta_{R,T}(t) &= R \cdot [t-T]^+ \\ &= 500 \text{ kbps} \cdot [t - 0.000272 \text{ s}]^+ \end{aligned}$$

6.3.4 Calculation of Delay Bounds

A central issue in CSMA/BA media access is the priority scheduling of CAN messages, i.e. how simultaneous transmissions of several CAN messages of different priority influence the transmission of a particular message. The arrival curves and the service curve are sufficient to compute the guaranteed worst-case delays at media access for CAN messages of all priority classes.

Let $x_p(t)$ denote the input data and $y_p(t)$ the output data of priority $p \in \{0, \cdots, P\}$ at time t. The iterative procedure of calculating upper delay bounds for all priorities starts with the highest priority class 0, considering input $x_0(t)$ and output $y_0(t)$:

$$d_0 \leq \sup_{t \geq 0} \{\inf\{\tau : \alpha_0(t) \leq \beta(t+\tau)\}\}$$

The arrival curve $\alpha_0(t)$ is a step function bounding the input $x_0(t)$. The service curve $\beta(t)$ is defined as presented beforehand:

$$\beta(t) = \beta_{R,T}(t) = 500 \text{ kbps} \cdot [t - 0.000272 \text{ s}]^+$$

CAN messages from the highest priority class 0 dominate all other priorities at simultaneous media access. Thus, only the worst-case non-preemption period of duration T contributes to the worst-case delay d_0.
Next, the maximum delay d_1 for the next lower priority class 1 must be determined. However, it would not be correct to apply the general service curve $\beta(t)$, as in the worst-case scenario, CAN messages of priority 1 will be served only after sending of CAN messages from class 0 is completed. In other words, the higher priority CAN messages detract CAN service capabilities from lower priorities by being sent first. This has to be reflected by adjusting the service curve $\beta(t)$ appropriately. Construction of the priority-class-dependent service curves follows the approaches of aggregate traffic modeling [17] and CPU task scheduling with preemption [57], [63].

For a reasonable application of aggregate scheduling, the service curve is required to be strict according to the definition given beforehand. For CAN, this can be guaranteed with respect to the actual technological realization of media access and data transfer: Assume there is a period with backlog in the system, i.e. several messages are waiting for service, ordered by priority. Non-preemption of transmission of a message applies only at the very beginning of servicing of the set of backlogged messages. Afterwards, messages are transfered continuously according to the priority, as soon as the bus becomes idle and no lower priority message can interrupt this process and cause additional, unexpected delay. Thus, the output of a CAN bus will never under-run the service guaranteed by the priority-dependent rate-latency service curve, as an idle bus never remains unemployed during a backlogged period.

In general, for servicing of CAN messages of priority class 1, the cumulative arrivals of messages from class 0 are subtracted from the service curve $\beta(t)$ according to:

$$\beta_1(t) := [\beta(t) - \hat{\alpha}_1(t)]^+$$

Even though $\hat{\alpha}_1(t) = \alpha_0(t)$ is a valid arrival curve for the cumulative input flow of class 0, constructing a service curve according to the formula above yields a service curve as shown in figure 6.5. For sure, this cannot be a valid cumulative service constraint, as the leaps in this saw-tooth-like curve would mean that the overall number of served data both *increases and decreases* along time, which is not possible for a *cumulative* service curve.

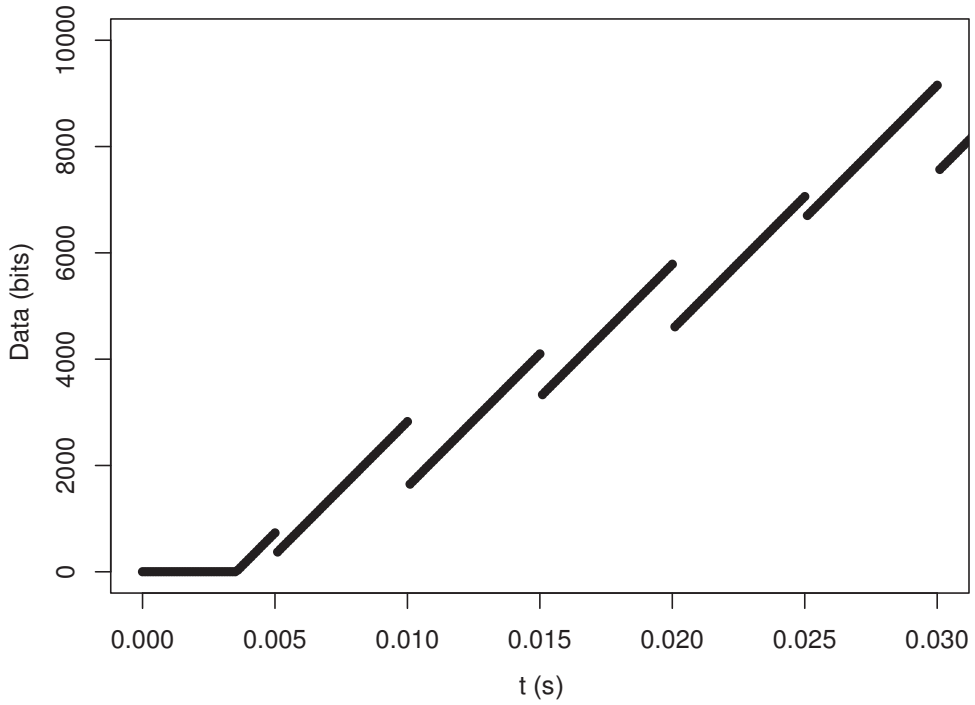


Figure 6.5: Nonmonotonic Service Curve

To overcome this, a linear function $\bar{\alpha}_p(t)$ is constructed as an upper bound for the step function arrival curve $\hat{\alpha}_p(t)$ of higher priority CAN messages:

$$\begin{aligned}\bar{\alpha}_p(t) &= p \cdot l + \frac{\sum_{i=0}^{p-1} C/c_i \cdot l}{C} \cdot t \\ &= p \cdot l + \sum_{i=0}^{p-1} \frac{l}{c_i} \cdot t \\ &= \sum_{i=0}^{p-1} \left(\frac{t}{c_i} + 1 \right) \cdot l\end{aligned}$$

The construction of the arrival curve $\bar{\alpha}_p(t)$ is based on the following considerations:

Without loss of generality, $\bar{\alpha}_p(t)$ is assumed to be a token bucket arrival curve. Thus, the vertical offset – the instantaneous burst b – and the slope of the curve – the rate r – have to be identified. The offset is obtained by adding all single offsets of the arrival curves $\alpha_p(t)$, i.e. it is assumed that all possible bursts occur

instantaneous after time $t = 0$. The slope is determined by summation of all slopes bounding the arrivals of higher priority classes, for which the boundaries are given as $\frac{C/c_i}{C} \cdot l = l/c_i$, because exactly C/c_i messages of length l with priority i are generated in each interval of length C.
As one can easily see, $\bar{\alpha}_p(t) \geq \hat{\alpha}_p(t)$, thus, $\bar{\alpha}_p(t)$ is an upper bound for the cumulative arrivals of higher priority messages that fulfills all preconditions for an arrival curve in Network Calculus.
Defining $b_p := p \cdot l$ and $r_p := \sum_{i=0}^{p-1}(l/c_i)$, $\bar{\alpha}_p(t)$ can be written in the typical token bucket form:

$$\bar{\alpha}_p(t) = b_p + r_p \cdot t$$

Using $\bar{\alpha}_p(t)$ instead of $\hat{\alpha}_p(t)$ for calculating $\beta_1(t)$ guarantees that a non-decreasing function and thus a valid service curve is obtained:

$$\beta_1(t) := [\beta(t) - \bar{\alpha}_1(t)]^+$$

Now, the maximum delay d_1 for CAN messages of priority 1 can easily be computed:

$$d_1 \leq \sup_{t \geq 0} \{\inf\{\tau : \alpha_1(t) \leq \beta(t+\tau) - \bar{\alpha}_1(t+\tau)\}\}$$

Continuing the iterative determination of worst-case delays from priority p to the next lower one $p+1$, a new service curve is built by diminishing the previous one by the arrival curve $\bar{\alpha}_{p+1}(t)$ for all frames with priority $0, 1, \ldots, p$. Hence, at each step of the iteration, only two kinds of priorities are considered: the messages of current priority and the sum of all higher priority CAN messages. All lower priority classes from $(p+1, \cdots, P)$ have no impact on the transmission of messages from priority p, besides maybe a single message just in service. However, this has already been modeled by the non-preemptive scheduling in the latency component T of $\beta_{R,T}(t)$.

The following procedure summarizes the approach for an iterative, priority-dependent calculation of upper delay bounds $\forall p \in \{0,1,\ldots,P\}$ and $t \in [0,\infty]$:

$$\begin{aligned}
\alpha_p(t) &= \left\lceil \frac{t}{c_p} \right\rceil \cdot l \\
\bar{\alpha}_p(t) &= b_p + r_p \cdot t \\
\beta_p(t) &= [\beta(t) - \bar{\alpha}_p(t)]^+ \\
&= \left[R[t-T]^+ - (b_p + r_p \cdot t)\right]^+ \\
&= [R(t-T) - (b_p + r_p \cdot t)]^+ \\
&= \left[(R - r_p) \cdot \left(t - \frac{RT + b_p}{R - r_p}\right)\right]^+
\end{aligned}$$

Again, a rate latency service curve is obtained with rate $R'_p := R - r_p$ and latency $T'_p := (RT + b_p)/(R - r_p)$. The input flow of priority p, together with the adjusted service curve $\beta_p(t)$, allow for a determination of the worst-case delay bound according to:

$$d_p \leq \sup_{t \geq 0} \{\inf\{\tau \geq 0 : \alpha_p(t) \leq \beta_p(t+\tau)\}\}$$

Due to the actual shape of the arrival curve $\alpha_p(t)$ and the service curve $\beta_p(t)$ in this case, this inequation can be solved geometrically by calculating the intersection point of $\beta_p(t)$ with the height of the first step of the arrival curve $\alpha_p(t)$, which is exactly $l = 136$ bits. This yields the following results:

$$\begin{aligned}
d_p &\leq \frac{l}{R'_p} + T'_p \\
&= \frac{l + RT + b_p}{R - r_p} \\
&= \frac{(p+2) \cdot l}{R - \sum_{i=0}^{p-1} (l/c_i)}
\end{aligned}$$

The application of this simple equation is sufficient to calculate the maximum delay d_p for each priority class p, given the data rate R and the message length l. The only input data needed for each priority class p is the cycle length c_p.

6.3.5 Exemplary Message Schedule

The following example illustrates the process of generating the arrival curves and determining the service curves and delays. Assume five different CAN priority classes with arbitrarily chosen cycle times as shown in table 6.2. All CAN messages from these priorities are standard CAN frames with a maximum length of 130 bits, therefore $l = 136$ bits. The results for worst-case communication delays for a CAN data rate of 500 kbps are shown in the rightmost column of table 6.2.

Table 6.2: Application of Network Calculus to Exemplary Message Schedule

Prio.	Cycle	$\bar{\alpha}_p(t)$		$\beta_p(t)$		
p	c_p (ms)	b_p (bits)	r_p (bps)	R'_p (bps)	T'_p (ms)	d_p (ms)
0	50	0	0	500000	0.272	0.544
1	10	136	2720	497280	0.547	0.820
2	100	272	16320	483680	0.844	1.125
3	20	408	17680	482320	1.128	1.410
4	30	544	24480	475520	1.430	1.716

The results are depicted in figure 6.6. In the leftmost column, the arrival curve $\alpha_p(t)$ for each priority class p is shown. The center column contains the step functions $\hat{\alpha}_p(t)$, depicted in black steps, and the upwards bounding linear function $\bar{\alpha}_p(t)$, drawn as a solid gray line. Since $\hat{\alpha}_p(t)$ is the superposition of individual step functions for the individual priority classes, the height of the steps in vertical direction varies for different points in time.
The rightmost column shows the arrival curve $\alpha_p(t)$ as a solid horizontal line and the corresponding service curve $\beta_p(t)$ as a dashed line. The horizontal distance between the thin vertical lines marks the geometrically determined maximum delay d_p as the intersection point of $\beta_p(t)$ and $l = 136$ bits.
Different scales are used for time and data axis in the third column, compared to the first two columns in this figure. Otherwise, the intersection points would not be visible due to the different orders of magnitude in temporal and data dimension of arrival curve and service curve.

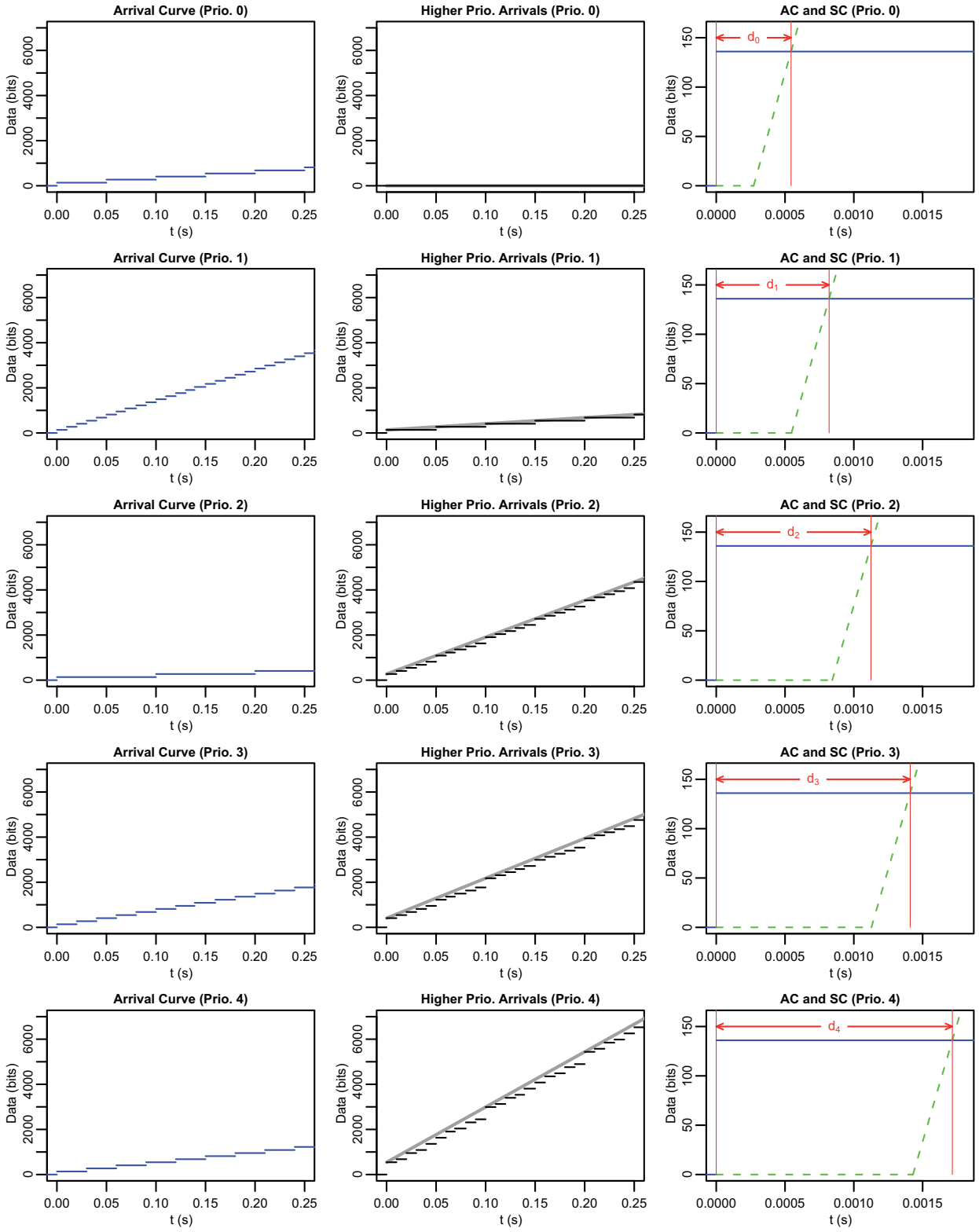


Figure 6.6: Graphical Visualization of Analysis Results for the Example

6.4 Application of Network Calculus to Overall In-Car Communication Topology

The application of worst-case analytics to local CAN data transmission is a valuable part of performance evaluation. However, a modern car comprises of several bus segments of various transmission technologies like CAN, FlexRay or MOST, which are typically connected by a gateway. Distributed, time- and safety-critical applications may rely on data which has to be transmitted from a *source bus* to a *destination bus*.
Like for local CAN data transmission, the method of Network Calculus is well-suited to derive guaranteed upper bounds for delays in a network-wide end-to-end communication scenario. The methodical approach and the required enhancements in modeling are presented in the following.

6.4.1 Methodical Approach

For an end-to-end communication scenario, it is assumed that several bus systems are connected to a gateway as depicted in figure 6.7. On contrast to the investigations of data transmission in a single CAN bus system, data dependencies may now span across borders of local bus segments, which is indicated in figure 6.7 by the labels *Source* and *Dest.* for an arbitrarily chosen pair of bus systems. Such a communication scenario requires more sophisticated analyses of data traffic in the network.

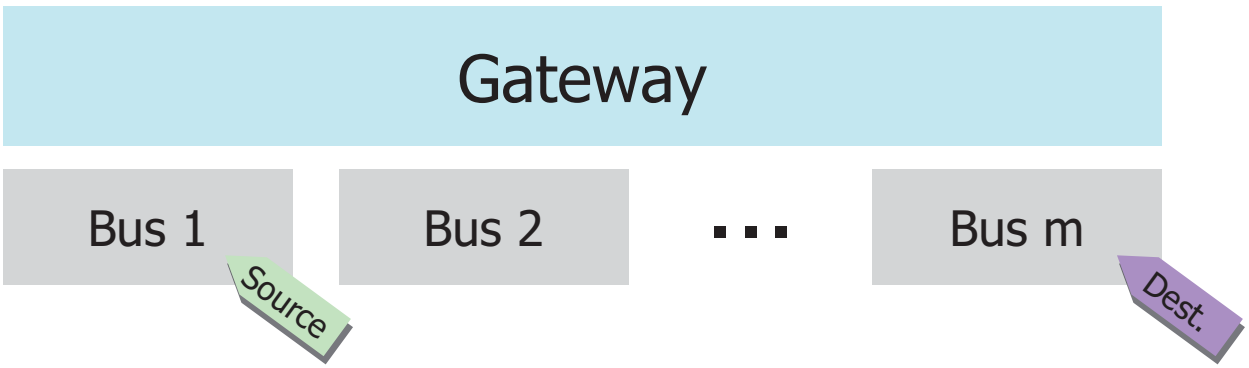


Figure 6.7: End-to-End Communication Scenario via Gateway

To obtain reasonable upper bounds, the complete flow of data traffic across the network must be considered when deriving worst-case end-to-end delays for certain messages. Assume data transmission along several sections of a communication system as depicted in figure 6.8.

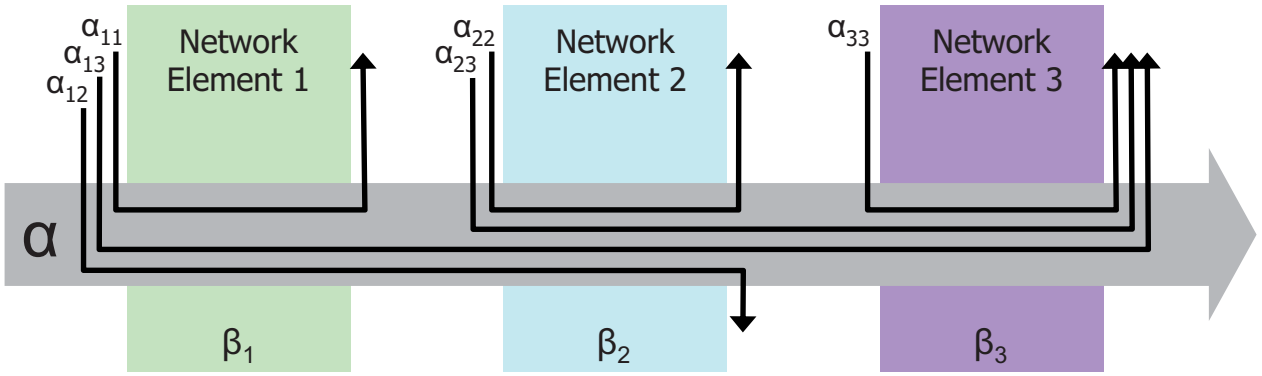


Figure 6.8: Arrival Curves in End-to-End Communication Scenario

In compliance to the nomenclature of Network Calculus, the single sections of the end-to-end transmission path are denoted as *Network Elements*. The traffic flow from source to destination is indicated by a broad, gray arrow, which is assumed to be bounded by an arrival curve α. Each network element offers specific service capabilities to the traffic flow, according to the service curves β_1, β_2 and β_3.
The traffic flow is compartmentalized into single flows with arrival curves $\alpha_{n,m}$, $n,m \in \{1,2,3\}$. These sub-flows join and leave the overall flow at the respective elements of the network, consume service capacity and thus contribute to transmission delays. Assuming an end-to-end communication scenario via network element 1, 2 and 3, the following holds for the sub-flows of traffic:

- $\alpha_{1,m}$ are arrival curves of sub-flows which attempt servicing at the first network element. Higher priority data in $\alpha_{1,m}$ consumes service capabilities from lower priority data at network element 1.
- $\alpha_{2,m}$ are arrival curves of sub-flows which demand service from network element 2. Depending on the strategy of processing, higher priority data may be served before lower priority data.
- $\alpha_{3,3}$ is the arrival curve of the sub-flow of data which is generated locally on network element 3. Higher priority data in $\alpha_{3,3}$ consumes local service capacity from network element 3, before lower priority data from network element 1, via network element 2, is served.

For derivation of reasonable end-to-end delay bounds, identifying which data from which sub-flow is relevant at which network element is a central issue in application of Network Calculus to a complete in-car communication system.

Identification of Traffic Flows

In figure 6.8, the general, flow- and sub-flow-based approach for worst-case analysis of end-to-end data transmission is illustrated. Substantiating it to an actual automotive network yields a scenario as depicted in figure 6.9. In dependence on the illustrations in figures 6.7 and 6.8, the communication system is composed of a source bus (*S*, *1* for network element 1), a gateway (*G*, *2* for network element 2) and a destination bus (*D*, *3* for network element 3). In addition, other bus systems are denoted by *O* and *O'*, respectively.
Again, α is the arrival curve of the complete end-to-end traffic flow from source to destination via the gateway, depicted as gray, arrow-headed line. The direction of the single sub-flows is indicated by the orientation of the black, arrow-headed lines. Source and destination are described by 2-tuples at the starting point and end point of the lines according to *(source, destination)*. Some examples:

- *(S,S)* in the sub-flow $\alpha_{1,1}$ resembles data which is generated on the source bus and kept local on the source bus.
- *(D,S)* in the sub-flow $\alpha_{1,1}$ stands for data which originates from the destination bus and is transfered via the gateway to the source bus.
- *(O,D)* in the sub-flow $\alpha_{2,3}$ depicts data being routed by the gateway from any other bus system to the destination bus.

Table 6.3 provides an overview, which sub-flows of data traffic in the network actually contribute to which arrival curve. Some sub-flows appear more than one time in the list, e.g. *(O,S)* contributes to $\alpha_{1,1}$ and $\alpha_{2,2}$. This is not a typo, moreover, data from these flows has to be considered at more than one network element, as clarified in the following.

A central issue in worst-case performance evaluation is to identify all traffic that possibly diminishes service capacity for a certain priority class on the one hand. On the other hand, an over-pessimistic modeling has to be avoided, e.g. by incorporating more traffic than theoretically present at the network elements.

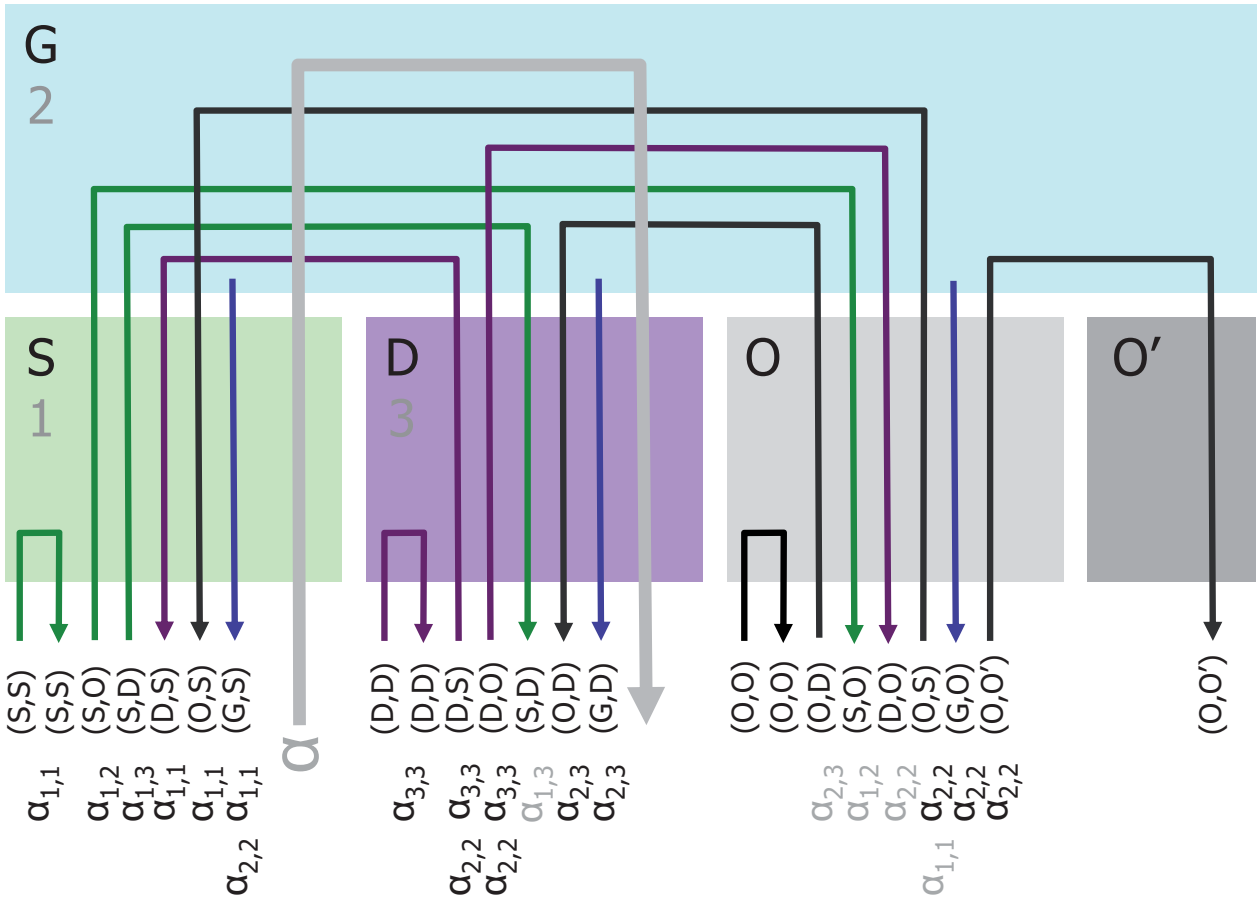


Figure 6.9: Traffic Flows for Worst-Case Analysis of Data Transfer

Table 6.3: Building Arrival Curves from Sub-Flows of Data Traffic

Arrival Curve	Sub-Flows
$\alpha_{1,1}$	(S,S) (D,S) (O,S) (G,S)
$\alpha_{1,2}$	(S,O)
$\alpha_{1,3}$	(S,D)
$\alpha_{2,2}$	(D,S) (D,O) (O,S) (O,O') (G,S) (G,O)
$\alpha_{2,3}$	(O,D) (G,D)
$\alpha_{3,3}$	(D,D) (D,S) (D,O)

For example, data exchange between other bus segments than source and destination – denoted by *(O,O')* – will surely neither effect bus arbitration at the source nor at the destination. Therefore, the flow from *(O,O')* does not contribute to the arrival curves $\alpha_{1,m}$ and $\alpha_{3,3}$, cf. table 6.3. However, data from *(O,O')* must be

routed by the gateway and at that point it may affect servicing of the flow from source to destination. That is why *(O,O')* is listed as part of the arrival curve $\alpha_{2,2}$ and has to be considered in network element 2.

Packetizing

Network Calculus is operating on flows of data, which is emphasized by the continuous-time functions modeling arrival curves and service curves. For local CAN communication, the ECUs are assumed to be direct receivers of data and hence the transmission delays can be computed straightforward, based on a continuous data flow into an ECU.
In a scenario of network-wide communication, a *store-and-forward* mechanism must be employed, as typically a data unit has to be received completely before it can be transfered to the next network element. For example, imagine two different CAN buses as source and destination. Data from the source bus arrives at a central gateway bit-by-bit with line speed. The gateway has to wait until the complete CAN frame was received from the source bus before the message can be routed. The continuous flow of bits cannot be transfered one-on-one continuously, but rather experiences a delay for storing all bits of one CAN frame and forwarding the complete message.
This constant store-and-forward delay is considered by application of a *packetizer* P^L. Assuming a sequence L of cumulative packet lengths, which is wide sense increasing, i.e. $(L(0) = 0, L(1), L(2), ...)$, such that

$$l_{max} = \sup_n \{L(n+1) - L(n)\}$$

is finite. In the following, l_{max} is set for *all* routed messages to 136 bits, which is the maximum size of a CAN frame[3]. An *L-packetizer* is the system that transforms an input $R(t)$ into $P^L(R(t))$ and fulfills for any real number x:

$$P^L(x) = \sup_{n \in \mathbb{N}} \{L(n) 1_{\{L(n) \leq x\}}\}$$

[3]FlexRay messages routed to a CAN bus, must be of at most 64 bit payload to be packageable in a standard CAN frame. Thus, data at the gateway or at the destination bus, originating from FlexRay, is assumed to be packaged as a 136-bit CAN frame already.

A packetizer ${P_n}^L$ is applied to the service curve of a network element n as follows:

$$\beta_n(t) \otimes {P_n}^L$$

The concatenation of $\beta_n(t)$ and ${P_n}^L$ leads to a service time prolongated according to the time span for storing and forwarding of a complete message. Without this important delay component, too optimistic upper bounds for end-to-end transmission delays would be achieved.

Modeling of FlexRay

Throughout this thesis, FlexRay communication is restricted to the static segment of a TDMA cycle. Thus, data transmission is strictly organized according to a TDMA communication schedule and no priorities have to be assigned in order to manage media access. Consequently, no arrival curves for higher priority data have to be constructed and accounted for effective servicing as done for CAN.

Arrival Curve Assuming a set of N FlexRay messages, a message i out of this set, $i \in \{0, 1, \cdots, N-1\}$, is of certain length l_i – including header, trailer and payload – and sent in individual cycles c_i. Hence, the cumulative arrivals of FlexRay messages are best resembled by individual arrival curves $\alpha_i(t)$, which are step functions bounding the single input flows $x_i(t)$.

Service Curve A FlexRay bus typically provides a data rate of 10 Mbps – the rate R in a service curve $\beta_{FR}(t)$, which is shown in figure 6.10.

Obviously, servicing at FlexRay depends on the message cycle c_i, the length l_i and the data rate R. In the worst-case, the very first message of type i has to wait a complete cycle time c_i before it can be transfered on the bus with rate R. This takes a transmission time of exactly l_i/R. The next message of the same type can be transmitted in the next time slot, which begins exactly $c_i - l_i/R$ time units after sending of the first message was completed. The same holds for the third, the fourth and the fifth message and so on.

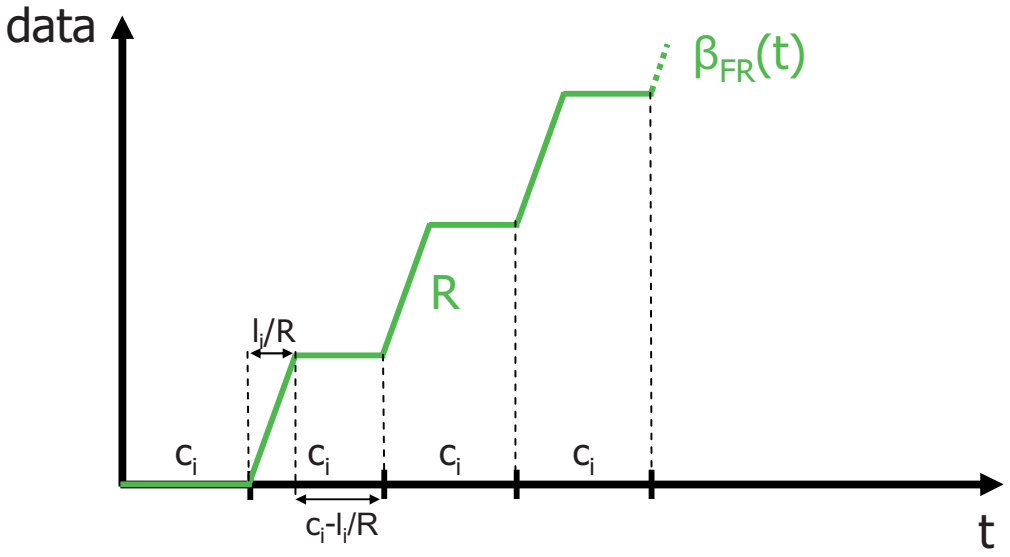


Figure 6.10: FlexRay Service Curve

Modeling of Central Gateway

As depicted in figures 6.8 and 6.9, the gateway typically relates to *Network Element 2* in worst-case analysis of network-wide communication. The analytical modeling approach for this element depends on the actual routing strategy. Basically, a gateway offers service capabilities in terms of routing functionality, i.e. receiving and sending of data from and to various connected bus systems. The routing process itself is assumed to be performed non-preemptive, with a constant rate that depends on the processor performance of the switching unit. This allows for determining the rate R_{gw} in a service curve for the gateway. The worst-case latency, a CAN or FlexRay message has to wait before it is routed, has to be derived according to the routing strategy.

Gateway Routing by Cyclic Polling Figure 6.11 shows a routing scenario according to a cyclic polling routing strategy. Data from source buses is received via dedicated in-ports at the gateway and stored for service according to FIFO. The switching unit cyclically checks the ports for newly arrived data and withdraws the head of queue. Once the message is taken out of the queue, the routing is performed, which means the message is sent to the out-port of one or several destination buses.

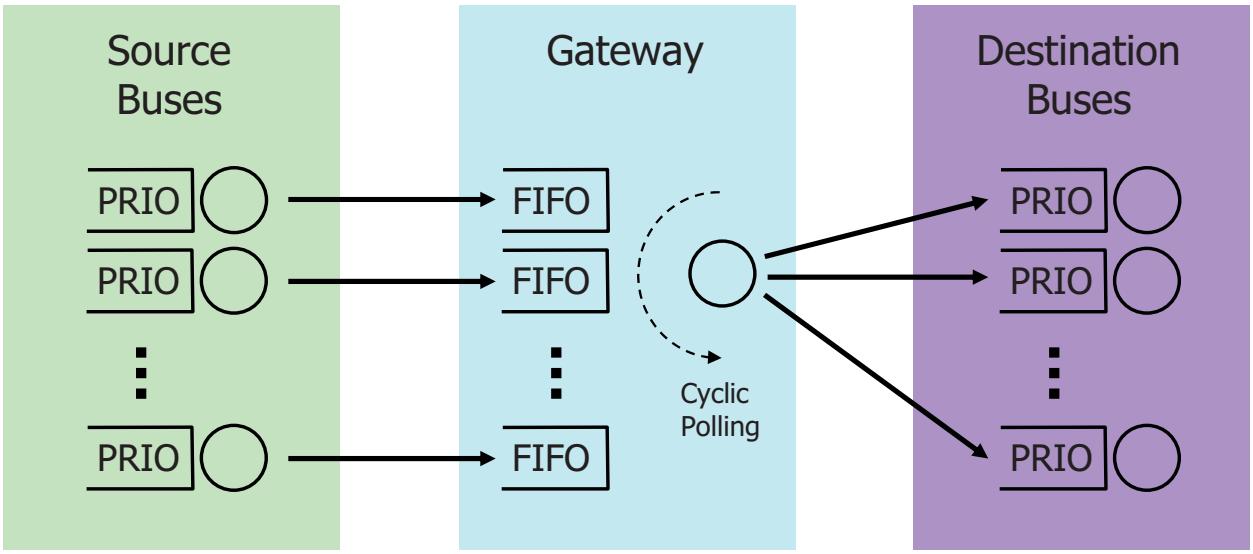


Figure 6.11: Routing by Cyclic Polling of In-Ports

The service curve $\tilde{\beta}_{gw}(t)$ for a gateway with a cyclic polling routing strategy with cycle time c_{gw} and a switching unit performance according to rate R_{gw} is depicted in figure 6.12.

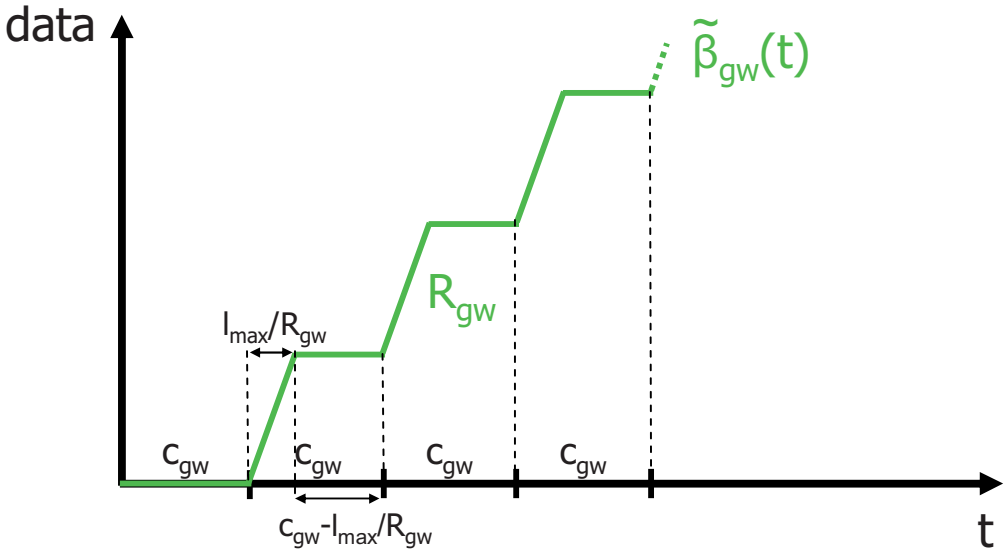


Figure 6.12: Gateway Service Curve for Cyclic Polling

The service curve for a cyclic polling routing strategy in the gateway is built in analogy to the FlexRay service curve. A first message of length l_{max}, arriving to a port, waits a worst-case time of c_{gw} and is processed in a time l_{max}/R_{gw} afterwards.

The next message can be routed in the next cycle, which begins $c_{gw} - l_{max}/R_{gw}$ time instances after processing of the first message, etc.

Gateway Routing by Strict Priority In figure 6.13, the routing strategy according to the priority of data is illustrated. Data from all source buses is stored in one central queue in the gateway and sorted by priority. A non-empty queue causes an interrupt, which triggers routing of the message with the currently highest priority in the queue.

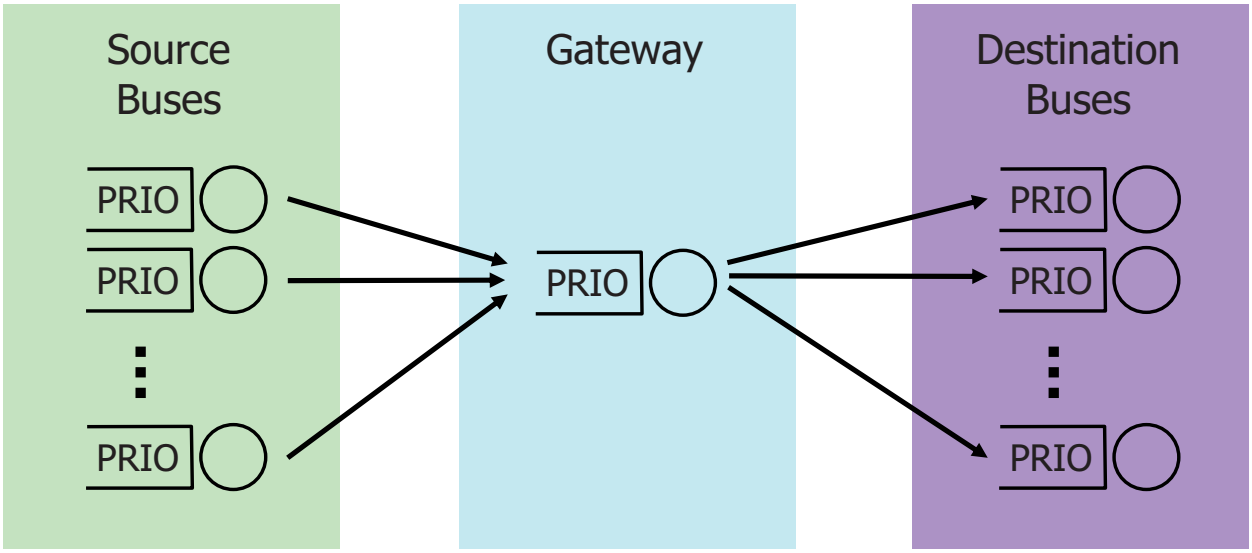


Figure 6.13: Interrupt-based Routing with Priority Queuing

In the worst-case, a message has to wait a latency $T = l_{max}/R_{gw}$ before routing. Servicing with gateway rate R_{gw} of a previous message of length l_{max} may just have started upon arrival of the current message and T captures non-preemptive servicing. For the highest priority message in the queue, the service curve $\bar{\beta}_{gw}(t)$ can be written as a rate-latency function:

$$\bar{\beta}_{gw}(t) = \beta_{R,T}(t) = R_{gw} \cdot [t - l_{max}/R_{gw}]^{+}$$

As shown in the following, for messages from the second-highest and all lower priority classes, an individual service curve must be constructed as done for CAN servicing. To this end, the initial rate-latency function $\bar{\beta}_{gw}(t)$ is diminished by the superposition of all higher priority arrivals at the gateway.

Worst-Case End-to-End Communication Analysis

The priority-dependent service curves for a priority class p are built individually for each network element along the end-to-end communication path. The following formulas summarize the determination of service curves for source bus, gateway and destination bus.

The service curve $\beta_{1,p}(t)$ for priority class p at the first network element, i.e. the source bus, is defined as:

$$\beta_{1,p}(t) = \left(\beta_1(t) - \sum_{j \in \{1,2,3\}} \hat{\alpha}_{1,j,p}(t) \right) \otimes {P_1}^L$$

If $\beta_{1,p}(t)$ violates the requirements of a service curve according to aggregate scheduling, a linear upper bound $\bar{\alpha}_{1,p}(t)$ for all higher priorities is constructed to achieve a valid service curve $\beta_{1,p}(t)$:

$$\beta_{1,p}(t) = (\beta_1(t) - \bar{\alpha}_{1,p}(t)) \otimes {P_1}^L$$

If the source bus is a CAN bus, this priority-dependent service curve complies to servicing in local CAN communication, except for the additional delay due to packetizing. If the source bus is a FlexRay bus, no arrivals of higher priority exist, i.e. $\bar{\alpha}_{1,p}(t) := 0$.

Servicing at the second network element – the gateway – is performed either according to cyclic polling with $\tilde{\beta}_{gw}(t)$ or strict priority with $\bar{\beta}_{gw}(t)$. As the service in the cyclic polling scenario is performed priority-independent, the service curve $\tilde{\beta}_{gw}(t)$ directly reflects worst-case servicing. $\beta_{2,p}(t)$ is determined for each priority class p as:

$$\beta_{2,p}(t) = \tilde{\beta}_{gw}(t) \otimes {P_2}^L$$

For routing from one central queue, strictly ordered by priority, the service capabilities of the gateway must be diminished by all data of higher priority. $\beta_{2,p}(t)$ is obtained as follows:

$$\beta_{2,p}(t) = \left(\bar{\beta}_{gw}(t) - \sum_{\substack{i \in \{1,2\} \\ j \in \{2,3\}}} \hat{\alpha}_{i,j,p}(t) \right) \otimes {P_2}^L$$

Applying a linear upper bound to the superposition of all higher priority arrivals yields:

$$\beta_{2,p}(t) = \left(\bar{\beta}_{gw}(t) - \bar{\alpha}_{2,p}(t)\right) \otimes {P_2}^L$$

At the third network element – the destination bus – the service curve is determined in analogy to the first network element, except for packetizing, as storage for further processing is not required at the end of the transmission path:

$$\beta_{3,p}(t) = \beta_3(t) - \sum_{i\in\{1,2,3\}} \hat{\alpha}_{i,3,p}(t)$$

To assure generation of a definition-compliant service curve, $\bar{\alpha}_{3,p}(t)$ bounds the sum of higher priority arrivals linearly:

$$\beta_{3,p}(t) = \beta_3(t) - \bar{\alpha}_{3,p}(t)$$

Again, if the destination bus is a FlexRay bus, no arrivals of higher priority exist, i.e. $\bar{\alpha}_{3,p}(t) := 0$.

From the individual, priority-dependent service curves $\beta_{1,p}(t)$, $\beta_{2,p}(t)$ and $\beta_{3,p}(t)$, the overall service curve $\beta_p(t)$ for a priority class p is obtained by min-plus convolution, i.e. by concatenation of the single service elements:

$$\beta_p(t) = \beta_{1,p}(t) \otimes \beta_{2,p}(t) \otimes \beta_{3,p}(t)$$

Once the overall service curve is determined, the worst-case end-to-end delay for priority class p in the network can be computed as:

$$d_p \leq \sup_{t\geq 0} \{\inf\{\tau \geq 0 : \alpha_p(t) \leq \beta_p(t+\tau)\}\}$$

Geometrically, calculating the intersection point of the height of the first step of the arrival curve $\alpha_p(t)$ and the end-to-end service curve $\beta_p(t)$ yields the worst-case end-to-end delay.

6.5 Discussion of Worst-Case Analysis Approach

Achieving guaranteed delay bounds for time-critical data transfer is a supremely valuable part of an all-encompassing performance evaluation of the underlying communication system. Although the analytical method of Network Calculus originates from research on classical Internet traffic, the mathematical basis and the modeling elements could be well adapted for modern automotive communication systems and transmission technologies. The adaption, enhancements and reasonable application studies remark the major contribution of this thesis regarding research on analytical network modeling.
Once the worst-case scenario in CAN data transmission was identified – bus-wide simultaneous sending of data from all ECUs – capturing CAN communication behavior was best done with commonly applied constructs, namely step functions for individual priorities, token bucket functions for higher priority arrivals and non-preemptive, constant rate service according to rate-latency functions.
Expanding the worst-case analysis to a complete in-car communication system, consisting of several buses and a gateway, required a sophisticated differentiation of the single flows of data within the network. At each network element, the distinction had to be made which data is relevant in terms of prioritized servicing. Therefore, generalized theoretical considerations, as illustrated in figures 6.8 and 6.9, were necessary to disentangle data flows in end-to-end transmission scenarios along several network elements.
Basically, the same mathematical modeling elements as for local CAN communication could be applied for end-to-end worst-case performance evaluation. Only minor changes were necessary, e.g. regarding the actual service strategy in the gateway or priority-independent TDMA data transmission at FlexRay.
The main benefit of worst-case analysis with Network Calculus is that basic communication data – message lengths, sending cycles and data rates or switching speeds – are sufficient to perform reasonable modeling at all stages for both local CAN communication and heterogeneous in-car networks.

7 Application Examples

This chapter presents detailed application studies for discrete event simulation and Network Calculus to issues of real-life in-car communication. The investigations were carried out in cooperation with engineers from the *Department of Safety Electronics at Audi, Ingolstadt*, who provided up-to-date communication data for performance evaluation of time-critical data transmission in various scenarios and safety system architectures.

7.1 Local CAN Bus Communication

For local CAN communication, a scenario was investigated, where a total of 19 ECUs are connected to the same CAN bus with a data rate of 500 kbps. The ECUs are broadcasting an overall number of 56 cyclic CAN messages to an error-free channel without packet-losses and retransmissions. Thus, some of the 19 ECUs are sending two or more different messages, i.e. with different CAN-ID and data content. Figure 7.1 provides a logarithmic-scaled presentation of the actually occurring cycle times for the 56 cyclic CAN messages.

7.1.1 Simulation Experiments

The simulation experiments were conducted with the AnyLogic™ simulation model as described in chapter 5. The actual number of ECUs – in this case 19 – determines the number of replicated `CAN ECU` active objects in the simulation. Each `CAN ECU` object is provided with its dedicated CAN messages (name, CAN-ID, cycle time) from an external file upon start of the simulation.

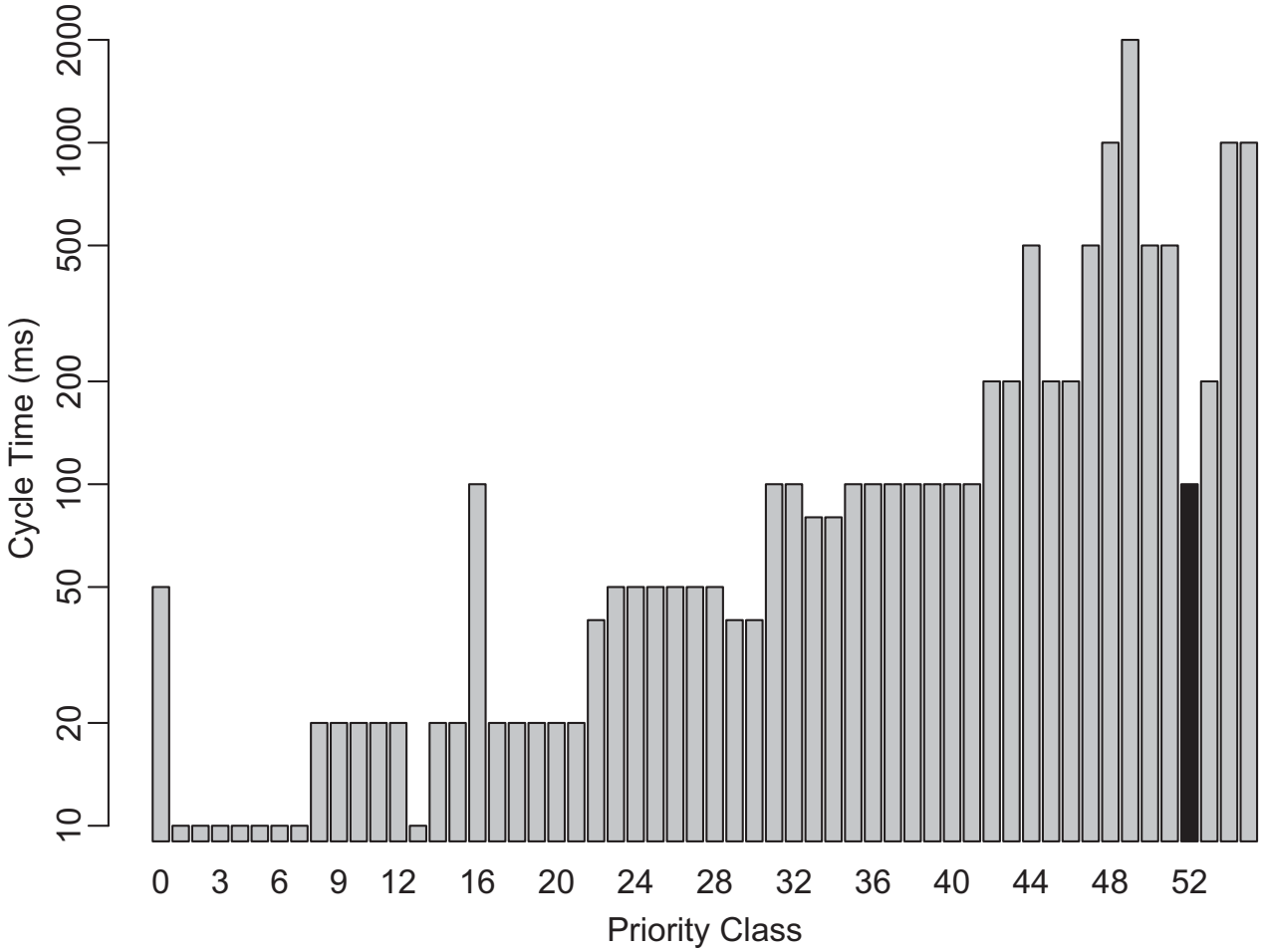


Figure 7.1: Cycle Times for 56 CAN Priority Classes

Typical Use-Case Simulation

From real-life measurements as presented in chapter 4, reasonable ranges and bounds for certain ECU parameters with impact on communication performance in use-case scenarios could be derived, namely for the controller startup duration and the frequency drift. Upon start of a simulation run, each replicated `CAN ECU` object is provided with a controller-dependent, realistic wake-up interval from which the actual point of time for ECU activation is sampled, and with a realistic value for μC frequency drift.

Worst-Case Simulation

In the worst-case simulation, all `CAN ECU` objects communicate simultaneously during the whole simulation run, i.e. the controller startup time is set to $t = 0$ and the frequency drift is also set to zero for each ECU.

7.1.2 Network Calculus

To compute upper bounds for worst-case delays analytically, the CAN-IDs – yielding a ranking in terms of priority classes – and the corresponding cycle times are sufficient.

7.1.3 Performance Evaluation Results

According to the actual approach, the performance evaluation results were obtained differently. For use-case simulation, stochastically independent replications were performed with a control mechanism aborting the simulation, if the relation between mean confidence interval and mean delay – calculated as the time difference between message creation and reception – for a particular priority class[1] falls below a threshold ε of 10% [36]. Actually, 119 replications, each one representing 10 seconds of real-life CAN communication, were necessary to achieve to predefined accuracy.
As no statistical variation plays a role in a worst-case communication scenario, worst-case simulation was performed in a single simulation run with global ECU startup time $t = 0$ and no frequency drift. Again, this simulation run represented 10 seconds of real CAN operation. Obviously, the expectable delays from a worst-case simulation must be larger or at most equal to the maximum delays in the typical use-case simulation scenario, where varying startup times are utilized. Analytical results for Network Calculus were computed according to the formulas derived for local CAN communication in chapter 6.

Table 7.1 provides the concrete numeric values for some priority classes p and allows for comparing them to the simulation results.

[1] Priority 52 was chosen due to the promising combination of low CAN-ID and rather fast cycle time, cf. black bar in figure 7.1.

Table 7.1: Numeric Results for Performance Evaluation of Local CAN Communication

		Network Calculus					Typical Use-Case Simulation			Worst-Case Simulation
		$\bar{\alpha}_p(t)$		$\beta_p(t)$						
p	c_p (ms)	b_p (bits)	r_p (bps)	R'_p (bps)	T'_p (ms)	d_p (ms)	Mean (ms)	Median (ms)	Max. (ms)	Max. (ms)
0	50	0	0	500000	0.272	0.544	0.354	0.296	0.544	0.544
1	10	136	2720	497280	0.547	0.820	0.432	0.481	0.816	0.816
2	10	272	16320	483680	0.844	1.125	0.633	0.641	1.088	1.088
3	10	408	29920	470080	1.157	1.447	0.428	0.415	1.360	1.360
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
27	50	3672	208760	291240	13.075	13.542	0.950	0.544	6.233	7.616
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
54	1000	7344	242692	257308	29.070	29.599	1.301	0.816	10.120	17.140
55	1000	7480	242828	257172	29.614	30.143	1.744	1.164	11.210	17.410

In figures 7.2 and 7.3, important statistics for the delays of each priority class are depicted, namely the range of the occurring delays as whiskers with minimum and maximum as short horizontal lines, the mean delays as black dots as well as the medians shown as wider horizontal lines and the 99%-quantiles as white triangles for all transmitted messages from the typical use-case simulation, together with the maximum delays from both the worst-case simulation as crosses and from Network Calculus as circles.

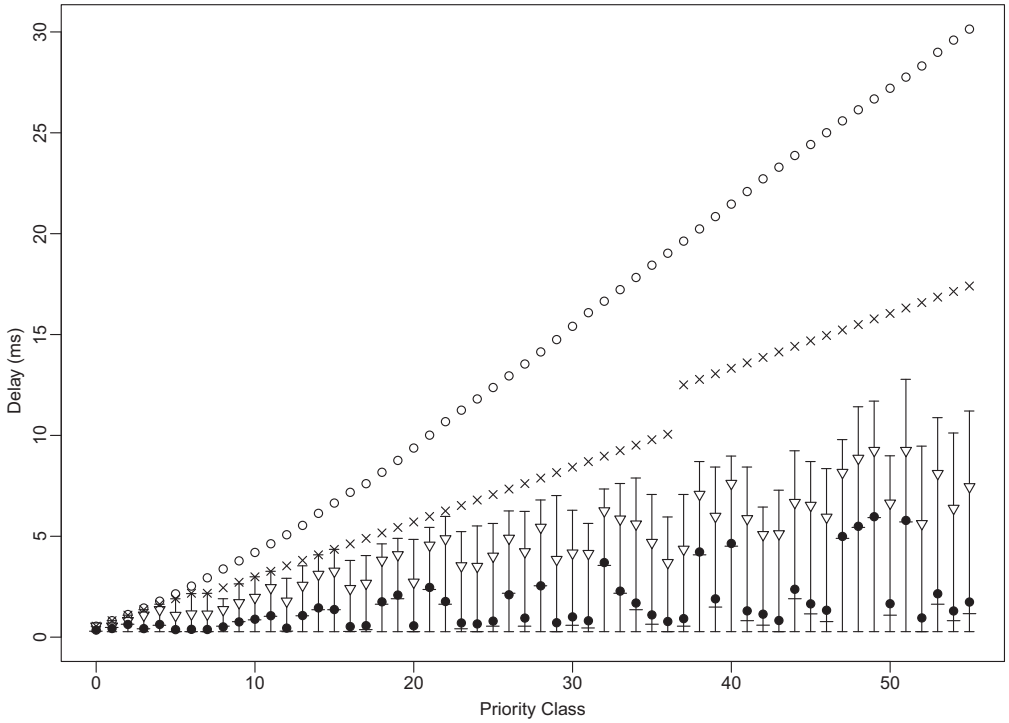


Figure 7.2: Performance Evaluation Results (Local CAN Bus, linear scale)

7.1.4 Comparison and Discussion

A comparison of use-case and worst-case simulation results with outcomes from analytical evaluation with Network Calculus yields first of all an implicit validation of the simulation experiments and in turn of the simulation model. No delay value from both simulation scenarios exceeds the hard analytical bounds, which is especially in the logarithmic-scaled representation in figure 7.3 very well visible. However, it can also be seen that the bounds provided by Network Calculus are not absolutely tight, which is caused by the approximation of both the arrivals and services as linear functions and the resulting rate reduction for the service

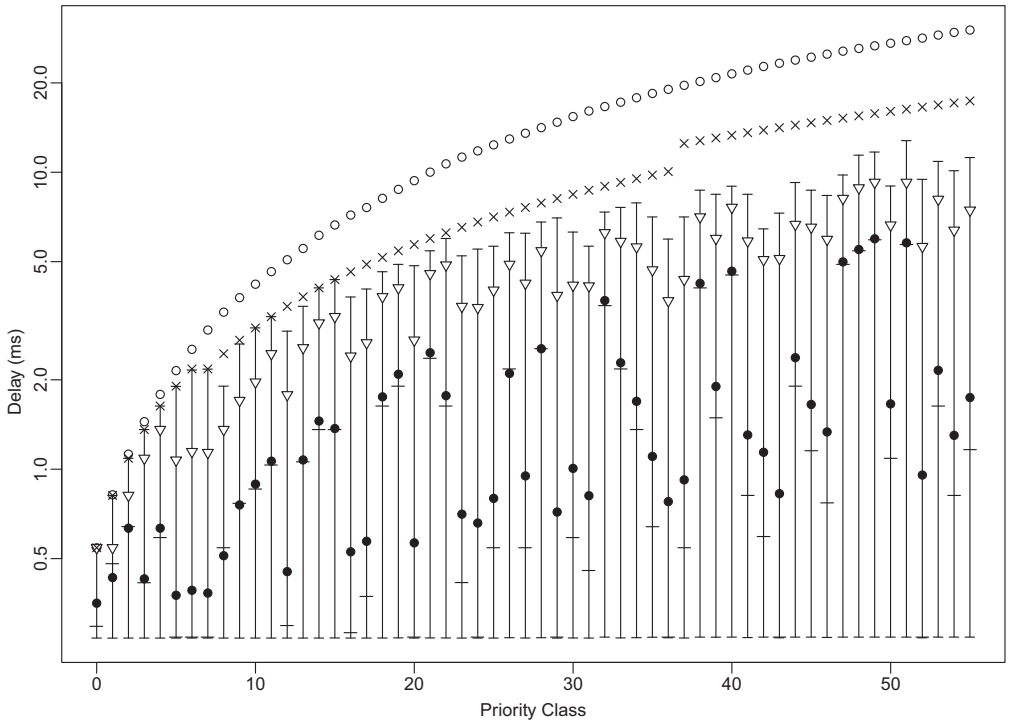


Figure 7.3: Performance Evaluation Results (Local CAN Bus, logarithmic scale)

of lower priority classes. In detail, the upwards bounding of the step-function-type arrival curves mainly contributes to this effect, as the linearization yields a slightly but inevitably increase of maximum input flow. Due to the superposition of higher priority arrivals, this overestimation intensifies, the lower the priority class. This explains, why the gap between simulation outcomes and analytical results increases from highest to lowest CAN-ID in use.
The discontinuity in the step height from priority 36 to 37 for maximum use-case simulation results in figures 7.2 and 7.3 is caused by the specific distribution of cycle times and priority classes as depicted in figure 7.1. Inspecting the simulation trace shows that the first message of priority class 37 is trying to arbitrate the bus at time $t = 0$. Until it can successfully be transmitted, messages of higher priority are sent first. Due to the shorter cycle times of several higher priority messages, some of them are sent multiple times before a message from priority class 37 can be sent for the first time, resulting in a higher, non-linear increase in the maximum delay. This also clarifies that a message of priority class p in some constellations has to wait more than p times the duration of one CAN message before winning bus arbitration.

7.2 Network-Wide Data Transmission

Network-wide end-to-end data transmission was investigated for a typical upper-class vehicle communication topology as depicted in figure 7.4. Four high-speed CAN buses CAN 1, CAN 2, CAN 3, CAN 4 and one FlexRay segment are connected to a central gateway. The respective number of ECUs connected to a bus segment is denoted in the figure, yielding a total of 48 ECUs in the communication system.

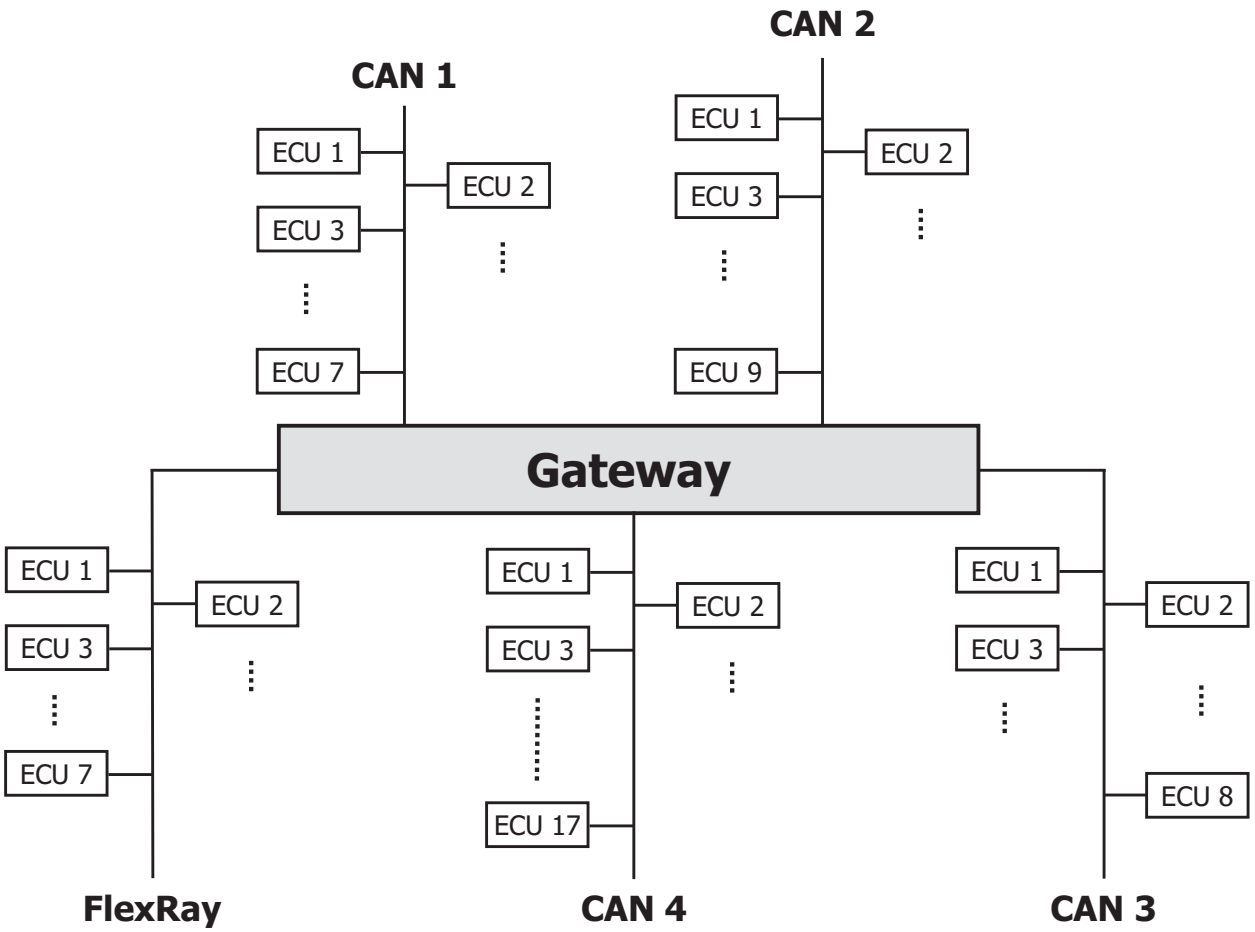


Figure 7.4: Network Topology for End-to-End Communication Scenario

Table 7.2 provides information on the communication parameters of the single buses, especially on the number of messages generated locally from the ECUs and of messages routed by the gateway from other segments to the respective bus. Again, error-free transmissions are assumed, i.e. no bit- or packet-errors and -losses on the communication channels and no retransmissions.

Table 7.2: Settings for Bus Segments in the Network

	CAN 1	CAN 2	CAN 3	CAN 4	FlexRay
Data Rate (bps)	500k	500k	500k	500k	10M
No. of ECUs	7	9	8	17	7
Messages from ECUs	30	33	15	54	140
Messages to Gateway	24	16	10	33	23
Messages from Gateway	31	68	40	25	38

For the gateway, the two operation modes – cyclic polling and strict priority – as presented in chapter 3 were considered according to the modeling approaches in chapters 5 and 6. The gateway processing rate R_{gw} was set to 13.6 Mbps, yielding an effective processing time of 10 μs for a 136-bit CAN frame. Consequently, the cycle time c_{gw} for cyclic polling was set to 50 μs for 5 connected buses. The gateway itself generates 15 relevant messages, for example for tasks of central network management or on-board diagnostics. These messages are routed to the CAN buses and to the FlexRay bus segment.

7.2.1 Simulation

On contrast to local CAN, a worst-case communication scenario which is valid at each of the single network elements cannot be identified directly for simulation of end-to-end data transmission. Imagine a message being transfered from one CAN bus to another via the gateway. If all ECUs at the single CAN buses are assumed to start operation at global time $t = 0$, the arbitration of the source bus will take the maximum possible time for this message. Considering the delay at media access, the transmission delay on the source CAN bus and the processing in the gateway, the message will access the destination bus at a time, when the initial burst of messages generated at time $t = 0$ has already been worked off to some extent. Of course, this reduces the arbitration delay for this particular message compared to a scenario, where all higher priority messages are present and preferentially served. Thus, only use-case simulation has been employed for network-wide data transmission. Again, the stochastically varying parameters are the ECU startup durations and the μC frequency drift. The replication mechanism for independent simulation runs is subject to the same simulation control as employed for simulation of local CAN data transmission.

The FlexRay ECUs are assumed to generate messages asynchronous to the TDMA schedule, yielding the maximum time gaps of asymptotically one message cycle time between generation and sending.

7.2.2 Network Calculus

The parameters for buses, messages and the gateway, as well as routing dependencies and communication paths, could be derived from the communication data provided by Audi. From these, the traffic flows were identified according to the Network Calculus application methodology as presented beforehand. The data rates and message lengths contributed to the generation of individual arrival curves, service curves and packetizer elements. The priority-dependent ranking of *all* messages transfered from one bus system to any other was accomplishable for the following reasons:

- CAN messages retain the assigned, network-wide unique CAN-ID, even in case of being routed to another CAN bus segment. Thus, the ID of a CAN message directly yields the respective priority class.
- For FlexRay messages which are routed to a CAN bus, the allocated CAN-ID for the destination bus also allows to assign a priority class to these messages.
- FlexRay messages which are not routed to other buses do not own a CAN-ID. As TDMA media access at FlexRay is performed independent of priorities and local FlexRay messages do not consume routing capacity in the gateway, these messages do not have to be considered in any priority ranking.

From table 7.2, it is obvious that a total of 272 messages are generated by ECUs on the five buses, plus an additional 15 messages from the gateway, yielding an overall number of 287 messages in the network. From the 140 FlexRay messages, only 23 are routed, the remaining 117 messages stay locally on the bus. Thus, 170 priority classes have to be introduced to establish a CAN-ID-based ranking of all relevant messages in the network. In a logarithmic-scaled representation, figure 7.5 depicts the CAN and FlexRay cycle times in which the 170 messages are generated.

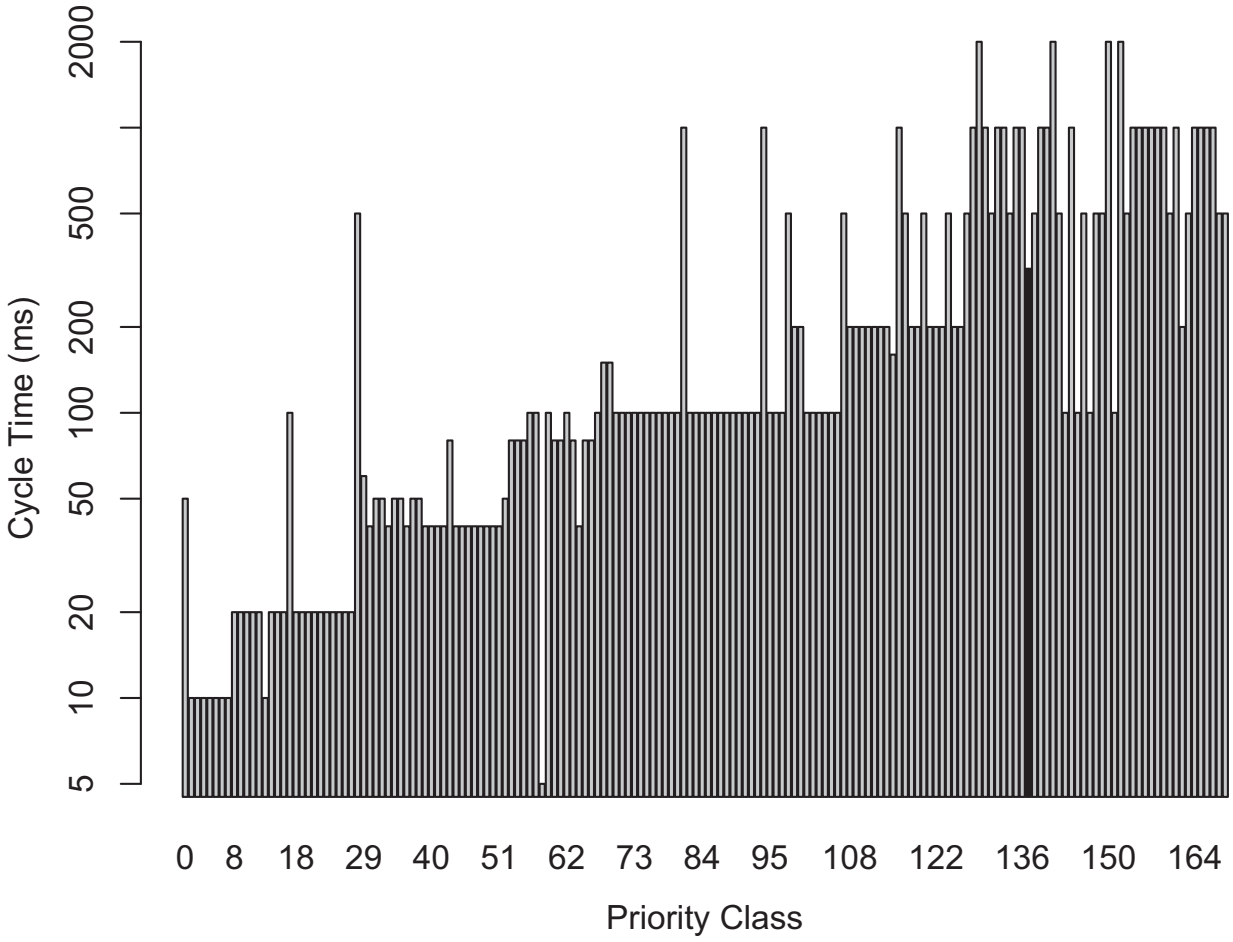


Figure 7.5: Cycle Times for all 170 Priority Classes in the Network

7.2.3 Performance Evaluation Results

The results for end-to-end performance evaluation were obtained for the given network topology of 4 CAN buses and 1 FlexRay for both a central gateway routing according to cyclic polling and strict priority.
As for local CAN communication, use-case simulation based upon independent replications. The end-to-end transmission delay of priority class 137 from FlexRay to CAN 2 was considered as abortion criterion, cf. black bar in figure 7.5. This message was particularly suited, as the rather slow cycle time at FlexRay yields a broad variety of durations between generation in the asynchronously operating FlexRay ECU and matching of the respective TDMA slot for sending. At the destination bus, priority class 137 is a rather low priority, leading to longer delays at CAN media access. A single replication runs at least until 10 messages from priority class 137 were transmitted.

For a network, where the gateway routes data by cyclic polling of in-ports, 268 replication were necessary until the relation between mean confidence interval and mean delay of priority class 137 messages under-ran a threshold ε of 10% in relative error. For a strict priority gateway routing mechanism, 350 independent replications were performed.
The application of Network Calculus was based on the formulas derived for network-wide worst-case analysis in chapter 6.
Applying both simulation and worst-case analysis, performance evaluation for all messages transfered between any pair of the 4 CAN buses and the FlexRay bus could be performed. In the following, exemplary outcomes for meaningful source and destination constellations will be shown and discussed thoroughly.

The graphical representation in figures 7.6 to 7.11 exactly follows the illustration as presented for local CAN communication in figures 7.2 and 7.3, e.g. the mean delays are shown as filled dots and the 99%-quantiles as white triangles. For each priority class, a pair of result values is given in each graphic. The left, gray-colored lines and symbols correspond to the evaluation outcomes for a strict priority routing and the right, black-colored properties comply with findings for cyclic polling routing. For reasons of better visibility – especially for results for higher priority classes lying at close quarters – all results are shown in a logarithmic scale.
The numerical values given in tables 7.3 to 7.8 are shown row-wise for the respective priority classes, for both routing according to cyclic polling (CP) and strict priority (SP). All results are denoted in milliseconds.

CAN-to-CAN Communication

In this scenario, messages are generated on a CAN bus and routed to another CAN. At both source and destination, the delay at media is expected to be related to the priority of the respective data. In figures 7.6 and 7.7, the results of simulative and analytical performance evaluation are illustrated graphically, tables 7.3 and 7.4 provide exemplary numerical values for highest, mid-range and lowest priority classes in these communication scenarios.

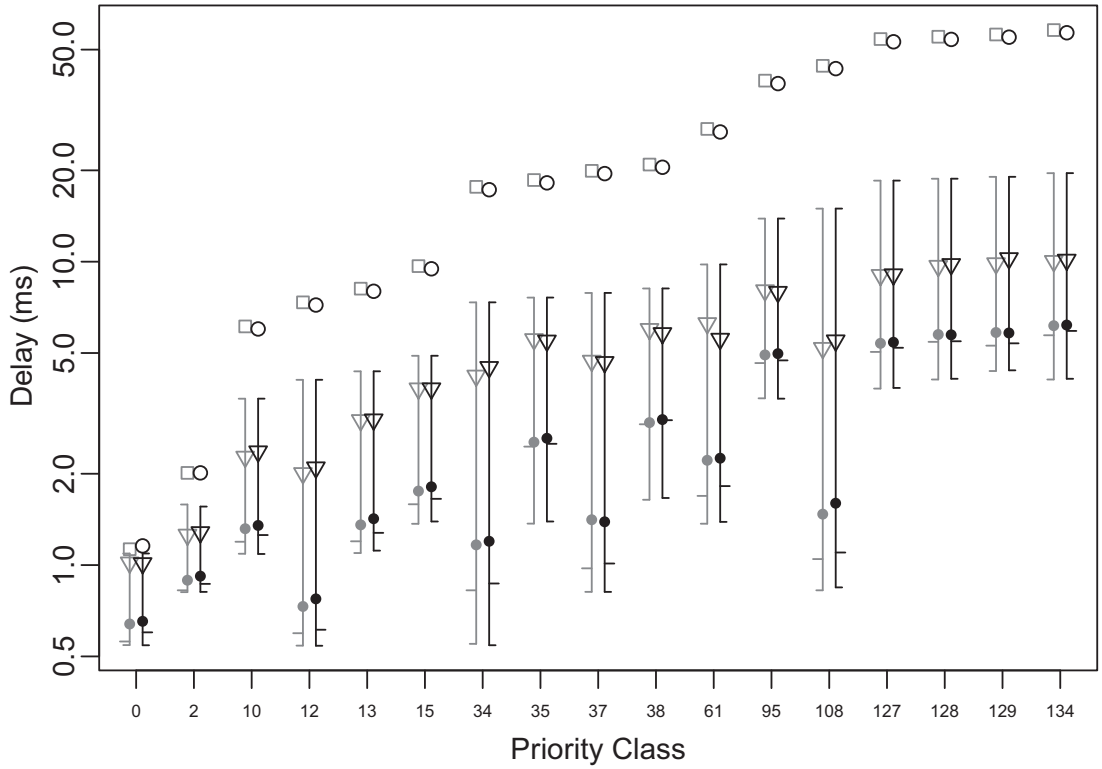


Figure 7.6: Performance Evaluation Results (CAN 1 → CAN 2)

Table 7.3: Numeric Results (CAN 1 → CAN 2, (ms))

Prio.		NC	Simulation				
		Max.	Min.	Max.	Mean	Median	q99
0	CP	1.158	0.544	1.092	0.653	0.600	1.014
	SP	1.128	0.545	1.092	0.639	0.560	1.019
2	CP	2.023	0.816	1.559	0.920	0.867	1.282
	SP	2.013	0.814	1.584	0.892	0.826	1.258
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
38	CP	20.613	1.664	8.160	3.019	3.003	5.808
	SP	20.899	1.642	8.160	2.951	2.912	5.994
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
134	CP	56.994	4.112	19.580	6.187	5.912	10.142
	SP	57.990	4.090	19.580	6.158	5.722	10.043

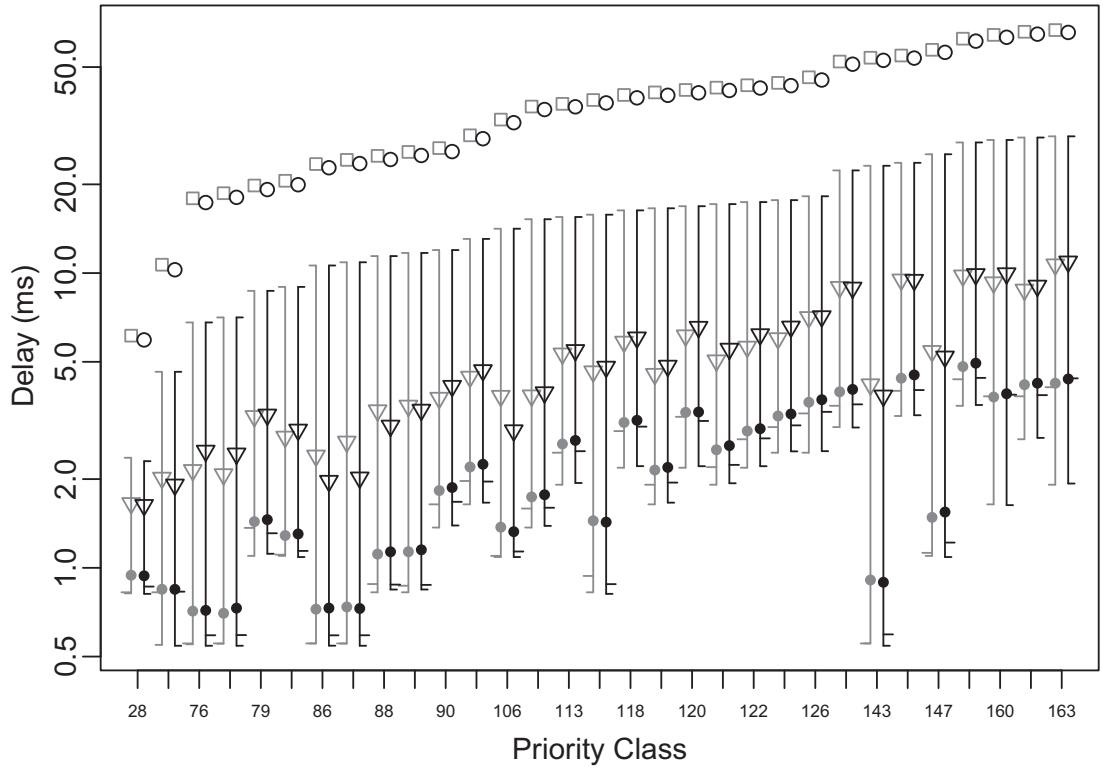


Figure 7.7: Performance Evaluation Results (CAN 4 → CAN 2)

Table 7.4: Numeric Results (CAN 4 → CAN 2, (ms))

Prio.		NC	Simulation				
		Max.	Min.	Max.	Mean	Median	q99
28	CP	5.939	0.816	2.301	0.940	0.864	1.632
	SP	6.133	0.819	2.363	0.944	0.826	1.656
52	CP	10.275	0.544	4.624	0.846	0.830	1.914
	SP	10.675	0.548	4.624	0.846	0.826	2.019
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
119	CP	40.145	1.664	16.590	2.193	1.946	4.844
	SP	41.008	1.642	16.590	2.148	1.914	4.544
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
163	CP	65.574	1.932	29.100	4.375	4.394	10.932
	SP	66.746	1.914	29.100	4.233	4.100	10.689

CAN-to-FlexRay Communication

Communication from a CAN bus to FlexRay captures CSMA/BA media access at the source bus and TDMA-based data transmission from the gateway to the destination. Again, for CAN media access the priority is assumed to be of impact, whereas bus access at FlexRay will mainly depend on the communication cycle in which TDMA slots are provided for a certain message. The results are denoted in both figures 7.8 and 7.9 and tables 7.5 and 7.6

FlexRay-to-CAN Communication

Data is generated by a FlexRay ECU, sent in the according TDMA time slot, received by the gateway and routed to a CAN destination bus. Again, the TDMA cycle time and the respective CAN priority class are considered to mainly contribute to the end-to-end transmission delay as depicted in figures 7.10 and 7.11 and illustrated numerically in tables 7.7 and 7.8.

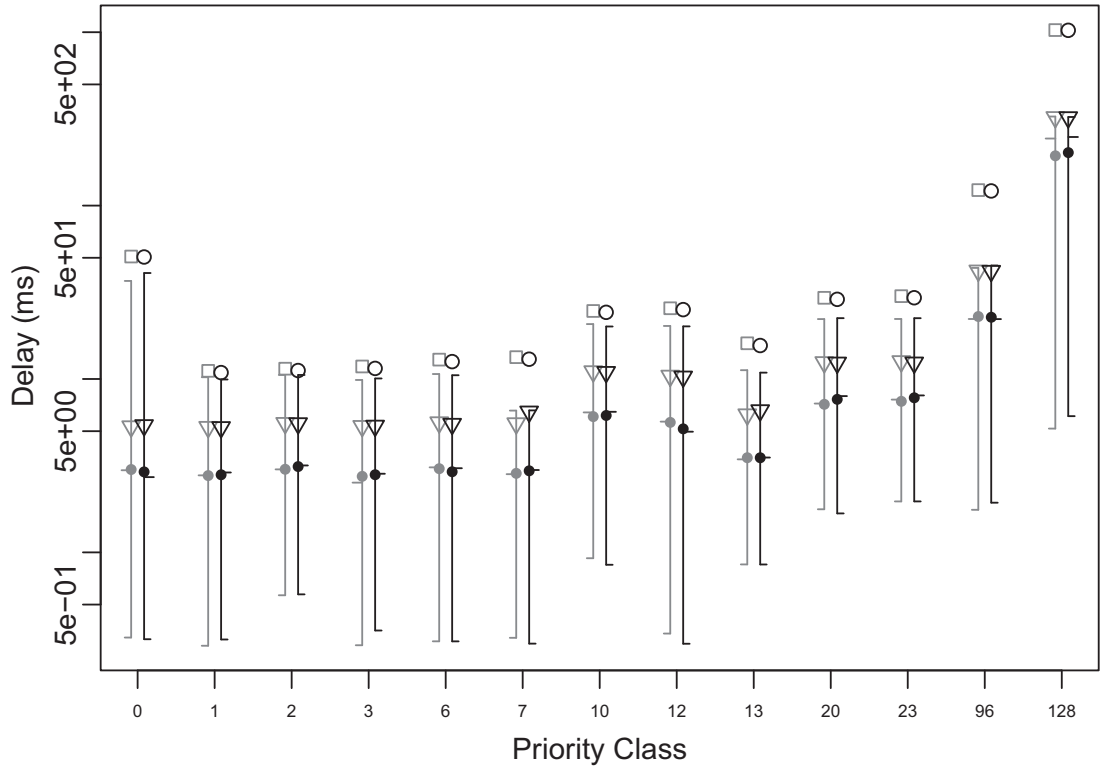


Figure 7.8: Performance Evaluation Results (CAN 1 → FlexRay)

Table 7.5: Numeric Results (CAN 1 → FlexRay, (ms))

Prio.		NC	Simulation				
		Max.	Min.	Max.	Mean	Median	q99
0	CP	50.886	0.316	40.840	2.916	2.716	5.407
	SP	50.856	0.322	36.770	3.009	2.976	5.323
1	CP	11.164	0.314	9.934	2.806	2.889	5.266
	SP	11.144	0.289	10.240	2.776	2.781	5.282
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
10	CP	24.621	0.848	20.080	6.166	6.467	10.956
	SP	24.672	0.925	20.730	6.074	6.415	11.061
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
128	CP	1027.860	6.104	324.300	202.400	248.700	322.778
	SP	1028.805	5.175	326.400	194.000	243.700	323.338

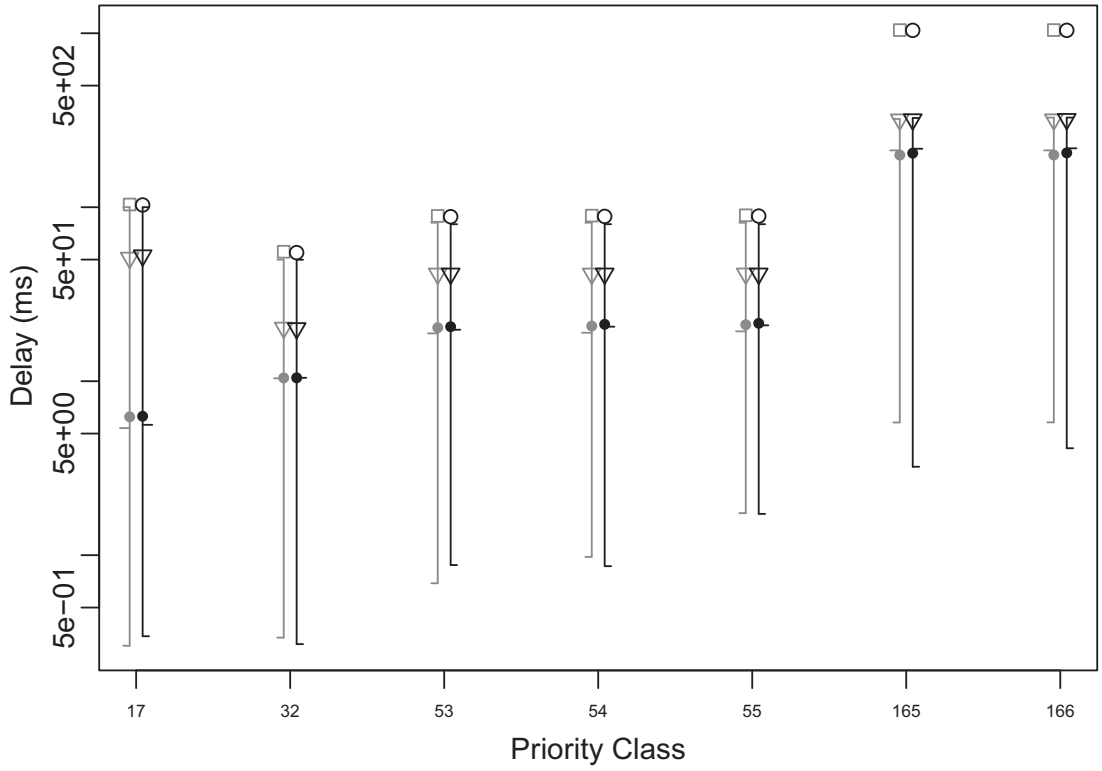


Figure 7.9: Performance Evaluation Results (CAN 2 → FlexRay)

Table 7.6: Numeric Results (CAN 2 → FlexRay, (ms))

Prio.		NC	Simulation				
		Max.	Min.	Max.	Mean	Median	q99
17	CP	103.500	0.342	100.000	6.296	5.613	52.784
	SP	103.622	0.301	100.100	6.244	5.378	51.174
32	CP	55.226	0.308	49.880	10.470	10.460	20.140
	SP	55.451	0.336	49.970	10.460	10.380	20.271
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
54	CP	89.050	0.864	79.850	21.260	20.580	41.482
	SP	89.470	0.978	81.490	20.780	18.990	41.455
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
166	CP	1041.492	4.118	327.600	205.600	218.000	320.765
	SP	1042.684	5.798	327.600	199.900	212.200	319.168

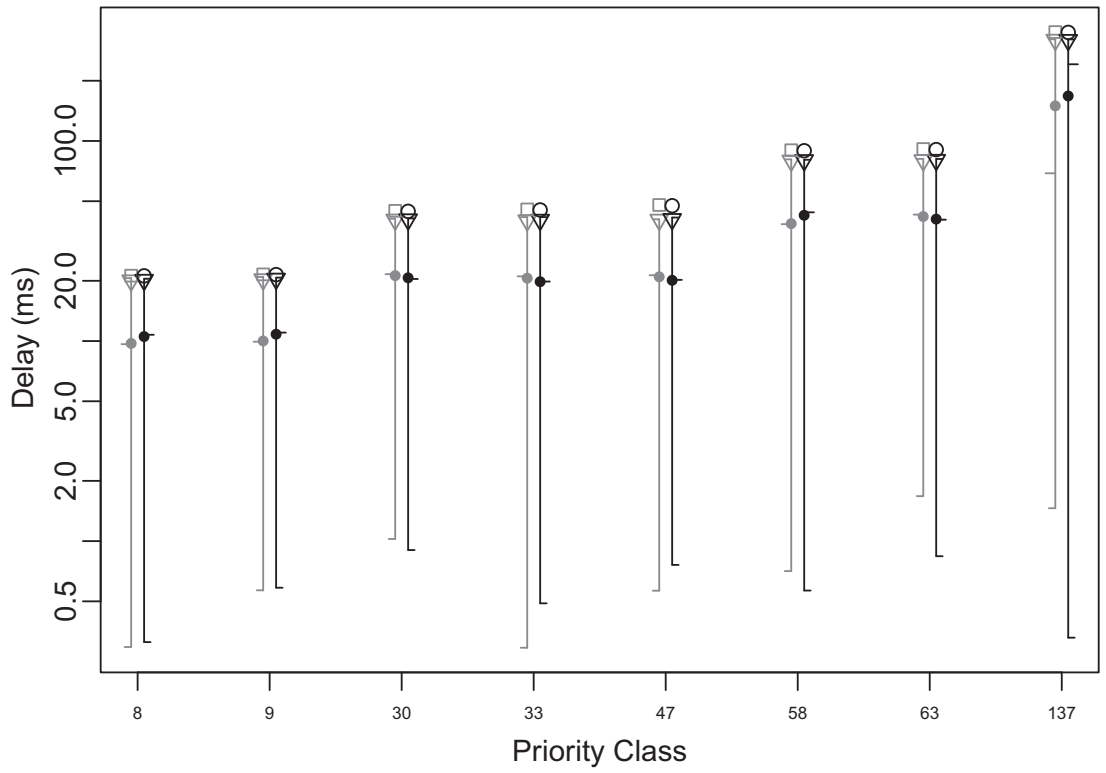


Figure 7.10: Performance Evaluation Results (FlexRay → CAN 2)

Table 7.7: Numeric Results (FlexRay → CAN 2, (ms))

Prio.		NC	Simulation				
		Max.	Min.	Max.	Mean	Median	q99
8	CP	21.194	0.313	20.480	10.540	10.750	20.197
	SP	21.225	0.296	20.630	9.738	9.646	20.065
9	CP	21.495	0.585	20.750	10.830	11.020	20.469
	SP	21.537	0.568	20.900	10.020	9.926	20.337
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
47	CP	47.520	0.761	41.430	20.140	20.220	40.667
	SP	47.868	0.565	40.610	20.960	21.330	40.370
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
137	CP	349.226	0.329	321.900	168.100	241.800	319.477
	SP	350.232	1.460	322.300	149.900	68.950	320.127

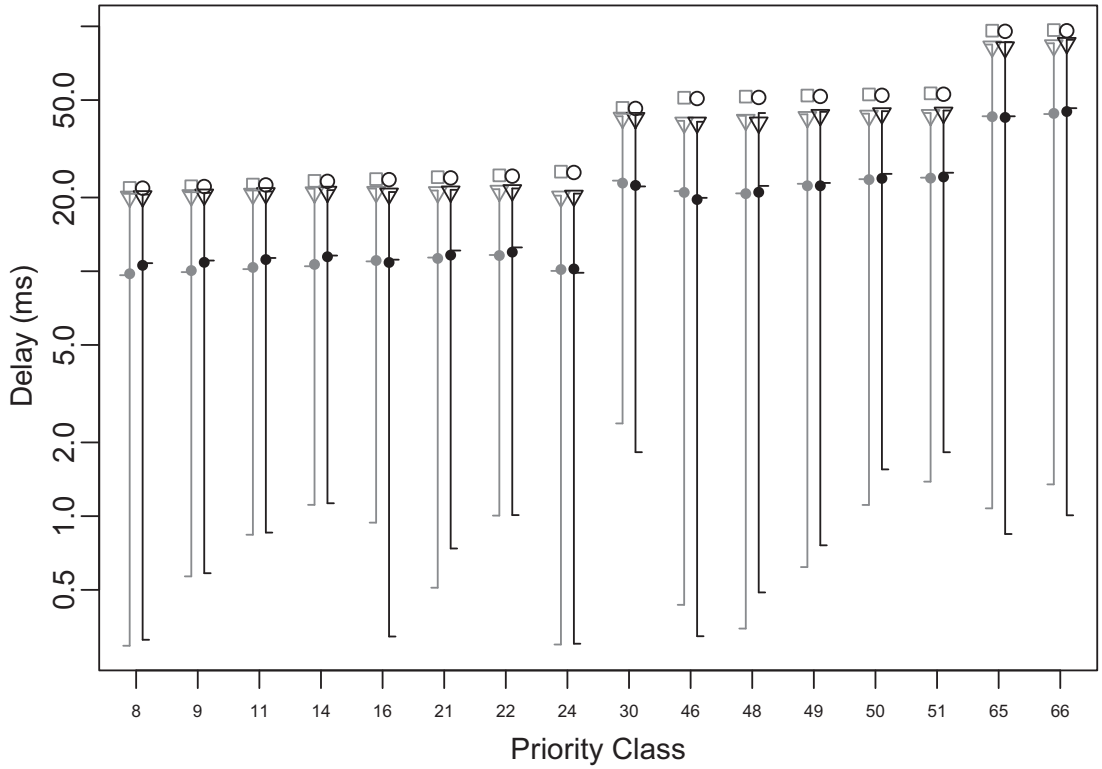


Figure 7.11: Performance Evaluation Results (FlexRay → CAN 3)

Table 7.8: Numeric Results (FlexRay → CAN 3, (ms))

Prio.		NC	Simulation				
		Max.	Min.	Max.	Mean	Median	q99
8	CP	21.857	0.313	20.570	10.580	10.790	20.224
	SP	21.888	0.296	20.900	9.773	9.634	20.146
9	CP	22.187	0.585	20.890	10.880	11.060	20.540
	SP	22.227	0.568	21.170	10.070	9.918	20.423
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
30	CP	46.237	1.825	43.610	22.430	22.200	42.203
	SP	46.441	2.391	42.930	22.920	23.440	42.312
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
66	CP	96.008	1.008	88.110	44.940	46.330	85.399
	SP	96.531	1.348	85.270	44.130	43.840	83.577

7.2.4 Comparison and Discussion

Several parameters contribute differently to end-to-end communication performance in the various scenarios. First and foremost, a cyclic-polling routing strategy appears to perform slightly better for a wider range of messages than routing according to a strict order of priority. For most lower priority classes, the FIFO ordering in the gateway buffer and a fix maximum timeout for service show more beneficial effects on delay than a priority-based routing, where higher classes are served preferentially. In turn, the higher priorities gain a bit more from the strict priority routing than from cyclic polling – at least as long as the worst-case timeout according to a ranking of priorities is lower than the gateway cycle time (cf. priority class 0 in figure 7.6). This is valid for both Network Calculus results and outcomes from use-case simulation throughout all scenarios.

For end-to-end transmission from one CAN bus to another, the priority class of data mainly contributes to the computed worst-case delays and the maximally observed simulation results. Again, the maximum results from both approaches are closer to each other for higher priority classes, whereas the gap between analytics and simulation increases for lower priorities due the necessary approximations and superpositions in Network Calculus as already discussed for local CAN communication[2]. However, analytics and simulation yield maximum values which monotonically increase from highest to lowest priority in use. The stochastic parameters in the simulation model were varied intensively, leading to a multitude of communication constellations in 268 and 350 independent replications. For example, the minimum end-to-end delay for class 143 is significantly lower than the minima values achieved for many higher priorities, cf. figure 7.7. Within the numerous replications, at least one message of this priority could be served very quickly, e.g. if source bus, gateway and destination bus were idle when the respective message had to be transfered. The 99%-quantiles of the simulation results emphasize that most of the messages in a use-case scenario experience delays of 5 - 10 milliseconds at most, the mean and median values are even significantly lower.

Data transmission from a CAN bus to FlexRay and vice versa is mainly dominated by the waiting time for the next transmission slot in the TDMA schedule. If the ECUs and the gateway operate asynchronous to the bus timing – which is up to

[2] In addition, no explicit worst-case could be *simulated* for end-to-end data transfer.

now standard, unless cost- and development-intensive AUTOSAR architectures are employed – end-to-end data transmission is considerably influenced by this asynchronism. From figures 7.8 to 7.11 and tables 7.5 to 7.8, it is obvious that the contribution of the priority class at routing and CAN media access is strongly dominated by the proportion of delay caused by the actual TDMA cycle time of the respective message. Consequently, a higher priority message does not necessarily experience a shorter end-to-end delay than a lower priority class. Moreover, the transmission time is widely independent of the initially assigned CAN-ID or priority class. Comparing priority classes 0 and 10 in figure 7.8 and table 7.5 or classes 17 and 54 in figure 7.9 and table 7.6 emphasizes this assumption.

The results are valuable information for the design of time-critical automotive applications. For example, imagine an ECU hosting functions for intelligent occupant protection, connected to CAN 2. Two environment sensors at CAN 1 and FlexRay provide essential information for this application, packaged in messages of global priority classes 0 and 47, respectively. For transmission from CAN 1 to CAN 2, class 0 data experiences delays of at most 1.158 ms and about half a millisecond in the mean, cf. figure 7.6 and table 7.3. Figure 7.10 and table 7.7 reveal that the transmission delay of priority 47 from FlexRay to CAN 2 is approximately 40 times higher both at maximum and on average, which is mainly caused by the delay at TDMA media access. Depending on the effective range and the actual design of the function, this significant difference in transmission delay would have to be considered thoroughly, e.g. by employing synchronization mechanisms for the respective sensor ECU at FlexRay or faster cycle times for information update and TDMA transmission.

Regarding TDMA and CSMA/BA media access and data transmission, the findings from the performance evaluation studies also comply with the results of the prototype measurements presented in chapter 4. The major benefit of the combined analytical and simulative performance evaluation is that for any message in an arbitrary end-to-end communication scenario both typically expectable delays and worst-case values are available. Instead of restricting the evaluation to data which is *currently* exchanged for time- and safety-critical applications, the consideration of the overall network traffic is meaningful for two reasons. First of all, the complete background traffic in a given network has to be incorporated anyway, as focusing only on the data of interest surely leads to overoptimistic performance measures. Once the background traffic is accounted for, it appears to be worthwhile to compute relevant performance characteristics for these data in one go. Messages,

which are not yet relevant in automotive safety applications of today, might contain data which is essential for innovative functions of tomorrow. Hence, any performance evaluation including the entirety of traffic yields reasonable studies for time-critical data transfer of nowadays. Over and above to this, a sound basis is provided for discussions about the expectable performance of innovations, which rely on additional data and thus enlarge the set of time-critical transmissions to be considered.

8 Conclusions and Future Work

8.1 Conclusions

The scope of this thesis was to enable an integrated performance evaluation of time- and safety-critical data transfer in automotive in-car networks, composed of CAN and FlexRay buses and a central gateway. From meaningful measurement studies on the communication system in a real-life prototype vehicle, several phenomena with a significant impact on the real-time capabilities of data transmission could be revealed.
The dynamic behavior of the communication system in the use-case, i.e. in normal operation mode, could be captured best with a discrete event simulation model based on the paradigms of UML-compliant statechart modeling and object oriented programming, employing the modeling tool AnyLogic™. The structure and behavior of basic components of the network – ECUs, messages, buses, gateway – were resembled closely. Meaningful simulation studies could be performed due to the reasonable parameterization and input modeling as obtained from measurements. The minima-, maxima-, mean- and quantile-related results are worthwhile information for the basic dimensioning of the system.
The worst-case of operation, which is for sure not negligible for data transmission with a scope on safety applications and occupant protection, was incorporated by analyses based on the method of Network Calculus. These worst-case analytics were applied to the field of automotive networking for the first time ever, requiring sophisticated considerations and enhancements to cope with constraints of in-car data transfer. The results are valuable information on the guaranteed maximum transmission delays and in turn on the lowermost expectable communication performance. Furthermore, the analytical outcomes allow for a sound validation of the maximum results from the simulation studies.
The three pillars of performance evaluation – measurements, simulation and analytics – have proven to be well-suited to facilitate comprehensive investigations

on timing aspects of in-car data transmission. The results from simulation and analytics were obtained for the entirety of data traffic, which assures a high level of trustworthiness. The complete background traffic is considered, which equips responsible engineers with an all-encompassing view on transmission performance on pretty much every message, data, etc. which is of interest for current or future networked safety applications.

8.2 Future Work

Automotive in-car electronics feature dynamics like currently few other technological application areas. Thus, there is plenty of room for future work on the topics investigated in this thesis. The measurement infrastructure can be enhanced in various directions, e.g. data collection inside the ECUs by employing ASAM XCP [2], determination of further performance measures like bit errors and packet losses or detailed inspection of data content for a signal- and situation-based evaluation of data traffic and behavior of the distributed system.
In the simulation model, real payload of varying size could be included in the CAN and FlexRay message objects, either as binary bit streams being coded in the sender and decoded in the receiver, or as semantical information on current states and signal values in a certain message. This would also support a more content-based simulation of communication and allows for extending the transmission chain, e.g. from data acquisition in a sensor to A/D conversion and coding, sending, receiving, decoding and, finally, triggering of applications, actuators, etc. Up to now, FlexRay data transmission is restricted to the static segment. Considering the dynamic segment in the simulation is an open issue to permit on-event sending beyond the static TDMA schedule in the model. Of course, other bus systems, like LIN or Ethernet, might be incorporated as well as wireless transmission technologies, like W-LAN, UMTS or RFID, in case the simulation model is to be coupled with Car-2-X environments, covering complete road traffic scenarios.
Regarding the worst-case analytics with Network Calculus, some efforts could be spent to achieve even tighter delay bounds. For example, an approximation for the step functions of message arrivals, which is closer than the linear upper bound, might be found and expressed mathematically. Processing the individual subflows, which contribute to the delays at the single network elements, according to

the approach of *Pay Multiplexing Only Once (PMOO)* [53] would be an alternative to obtain more optimistic end-to-end delay bounds. Just as for simulation, the dynamic segment of FlexRay may also be considered in worst-case analytics, where the priority-based media access in the dynamic part poses some interesting questions on the worst-case on-event transmission times of the individual FlexRay priorities. The concept of *Stochastic Network Calculus* [26] could be applied to determine guaranteed, quantile-related thresholds for communication performance. Finally, computing guaranteed upper bounds for the backlog in the system might be valuable information for dimensioning, e.g. of the buffers in the gateway or in the individual ECUs.

Bibliography

[1] A. Albert and W. Gerth. Evaluation and Comparison of the Real-Time Performance of CAN and TT-CAN. In *9th International CAN Conference (iCC)*, October 2003.

[2] ASAM - Association for Standardisation of Automation and Measuring Systems. ASAM MCD-1 XCP: The Universal Measurement and Calibration Protocol Family. http://www.asam.net/.

[3] AUDI AG. Audi Insassenschutz - Passive Systeme, Selbststudienprogramm 410.

[4] AUDI AG, Department for Safety Electronics. Data Sheets on Airbag Control Systems.

[5] AUTOSAR Consortium. Automotive Open System Architecture (AUTOSAR). http://www.autosar.org.

[6] P. Castelpietra, S. Ye-Qiong, F. Simonot-Lion, and M. Attia. Analysis and Simulation Methods for Performance Evaluation of a Multiple Networked Embedded Architecture. *IEEE Transactions on Industrial Electronics*, 49:1251–1264, 2002.

[7] CoCar Consortium. CoCar - Cooperative Cars, Project Report on UMTS-based Car2X Approaches. 2008. http://www.cocar.org.

[8] Condalo Ltd. Condalo Data Logger CCO DLIII Data Sheet and Manual. http://www.condalo.de/pid130.html.

[9] R. L. Cruz. A Calculus for Network Delay, Part I: Network Elements in Isolation. *IEEE Transactions on Information Theory*, 37:114–131, 1991.

[10] R. L. Cruz. A Calculus for Network Delay, Part II: Network Analysis. *IEEE Transactions on Information Theory*, 37:132–141, 1991.

[11] A. Davare, M. DiNatale, and Q. Zhu. Period Optimization for Hard Real-time Distributed Automotive Systems. In *44th IEEE/ACM Design Automation Conference (DAC)*. ACM, June 2007.

[12] R. I. Davis, A. Burns, J. R. Bril, and J. J. Lukkien. Controller Area Network (CAN) Schedulability Analysis: Refuted, Revisited and Revised. *Real-Time Systems*, 35:239–272, 2007.

[13] R. Ernst and K. Richter. Real-Time Analysis as a Quality Feature: Automotive Use-Cases and Applications. In *Embedded World Conference*, February 2006.

[14] K. Etschberger. *Controller Area Network*. Carl Hanser Verlag, München, Germany, second edition, 2000.

[15] Faurecia Industries. Final Report for SAVE-U (Sensors and System Architecture for Vulnerable Road Users Protection). 2005. http://www.save-u.org.

[16] J. Ferreira, A. Oliveira, P. Fonseca, and J. A. Fonseca. An Experiment to Assess Bit Error Rate in CAN. In *The 3rd International Workshop on Real-Time Networks (RTN)*, 2004.

[17] M. Fidler and V. Sander. A Parameter-based Admission Control for Differentiated Services Networks. *Computer Networks*, 44:463–479, 2004.

[18] FlexRay Consortium. *FlexRay Communications System Protocol Specification*. Version 2.1 edition, 2005.

[19] A. Hamann, R. Racu, and R. Ernst. Formal Methods for Automotive Platform Analysis and Optimization. In *Future Trends in Automotive Electronics and Tool Integration Workshop (DATE Conference)*, March 2006.

[20] T. Herpel and R. German. A Simulation Approach for the Design of Safety-Relevant Automotive Multi-ECU Systems. In *4th International Conference on System of Systems Engineering*. IEEE, June 2008.

[21] T. Herpel, K.-S. J. Hielscher, U. Klehmet, and R. German. Stochastic and Deterministic Performance Evaluation of Automotive CAN Communication. *Computer Networks 53*, pages 1171–1185, 2009.

[22] T. Herpel, B. Kloiber, R. German, and S. Fey. Assessing the CAN Communication Startup Behavior of Automotive ECUs by Prototype Measurements. In *International Instrumentation and Measurement Technology Conference (I^2MTC)*. IEEE, May 2009.

[23] T. Herpel, B. Kloiber, R. German, and S. Fey. Routing of Safety-Relevant Messages in Automotive ECU Networks. In *IEEE 70th Vehicular Technology Conference*. IEEE, September 2009.

[24] Institut für Datentechnik und Kommunikationsnetze, Universität Braunschweig. SymTA/S - Symbolic Timing Analysis for Systems. http://www.ida.ing.tu-bs.de/forschung/projekte/symtas/.

[25] International Standard ISO 11898. *Road vehicles - Interchange of Digital Information - Controller Area Network (CAN) for High Speed Communication*. International Organization for Standardization (ISO), first edition, 1994. ISO Reference Number ISO 11898:1993(E).

[26] Y. Jiang and Y. Liu. *Stochastic Network Calculus*. Springer Verlag, 2008.

[27] U. Klehmet, T. Herpel, K.-S. J. Hielscher, and R. German. Worst Case Analysis for Multiple Priorities in Bitwise Arbitration. In *Leistungs-, Zuverlässigkeits- und Verlässlichkeitsbewertung von Kommunikationsnetzen und verteilten Systemen (MMBnet)*, pages 27–35, September 2007.

[28] U. Klehmet, T. Herpel, K.-S. J. Hielscher, and R. German. Delay Bounds for CAN Communication in Automotive Applications. In *14th GI/ITG Conference on Measurement, Modelling and Evaluation of Computer and Communication Systems (MMB)*, pages 157–171. VDE Verlag, April 2008.

[29] U. Klehmet, T. Herpel, K.-S. J. Hielscher, and R. German. Real-Time Guarantees for CAN Traffic. In *IEEE 67th Vehicular Technology Conference*, pages 3037–3041. IEEE, May 2008.

[30] B. Kloiber. Messung, Analyse und Bewertung der Echtzeitfähigkeit vernetzter Fahrzeugsicherheitskomponenten. Diploma thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg, Lehrstuhl Informatik 7 (Rechnernetze und Kommunikationssysteme), Erlangen, Germany, 2009.

[31] K. H. Kraft. Simulation von CAN-Bus-Komponenten und -Übertragungsnetzen. In *GMM Fachbericht Analog 2002 - Entwicklung von Analogschaltungen mit CAE Methoden*. VDE Verlag, 2002.

[32] J. Krakora and Z. Hanzalek. Verifying Real-Time Properties of CAN Bus by Timed Automata. In *FISITA 2004 - World Automotive Congress*, May 2004.

[33] F. Kramer. *Passive Sicherheit von Kraftfahrzeugen*. Vieweg + Teubner, Wiesbaden, Germany, third edition, 2009.

[34] J. Langheim. CARSENSE - New Environment Sensing for Advanced Driver Assistance Systems. In *IEEE Intelligent Vehicle Symposium*, pages 89–94, 2001.

[35] A. M. Law. ExpertFit - Distribution Fitting Software. http://www.averill-law.com.

[36] A. M. Law and W. D. Kelton. *Simulation Modeling and Analysis*. McGraw-Hill, Boston, MA, USA, third edition, 2000.

[37] W. Lawrenz, editor. *CAN Controller Area Network*. Hüthig Verlag, Heidelberg, Germany, fourth edition, 2000.

[38] J.-Y. Le Boudec and P. Thiran. *Network Calculus*. Springer Verlag LNCS 2050, 2001.

[39] A. Leon-Garcia and I. Widjaja. *Communication Networks*. McGraw-Hill, New York, NY, USA, 2004.

[40] M. Meinecke. Approach for Protection of Vulnerable Road Users Using Sensor Fusion Techniques. In *International Radar Symposium*, 2003.

[41] U. Meis and R. Schneider. Radar Image Acquisition and Interpretation for Automotive Applications. In *IEEE Intelligent Vehicle Symposium*, pages 328 – 332. IEEE, 2003.

[42] MOST Cooperation. *MOST Media Oriented Systems Transport*. Revision 2.4 edition, 2005.

[43] N. Navet and Y.-Q. Song. Validation of In-Vehicle Real-Time Applications. *Computers in Industry*, 46:107–122, 2001.

[44] L. D. Ortega, K. H. Kraft, and L. Claus. Untersuchung von passiven FlexRay-Sternkopplern. *ELEKTRONIK automotive*, 7:64–67, 2007.

[45] R. Pallierer, M. Horauer, M. Zauner, A. Steininger, E. Armengaud, and F. Rothensteiner. A Generic Tool for Systematic Tests in Embedded Automotive Communication Systems. *Embedded World Conference*, February 2005.

[46] PSI 5 Consortium. PSI 5 - Peripheral Sensor Interface for Automotive Applications, Technical Specification, 2007. http://www.psi5.org.

[47] R. Racu, A. Hamann, and R. Ernst. Sensitivity Analysis of Complex Embedded Real-Time Systems. *Real-Time Systems*, 39:31–72, 2008.

[48] K. Reif. *Automobilelektronik*. Vieweg + Teubner, Wiesbaden, Germany, third edition, 2009.

[49] P. Richardson, L. Sieh, A. Elkateeb, and P. Haniak. Real-time Controller Area Networks (CAN) - Managing Transient Surges. *Integrated Computer-Aided Engineering*, 9:149–165, 2002.

[50] Robert Bosch GmbH. CAPS - Combined Active and Passive Safety. http://rb-k.bosch.de/de/sicherheitkomfort/fahrsicherheit/caps.

[51] J. Roberts, U. Mocci, and J. Virtamo. *Broadband Network Traffic - Performance Evaluation and Design of Broadband Multiservice*, volume 1155 of *Lecture Notes in Computer Science*. Springer Verlag, December 1996.

[52] S. Samii, S. Rafiliu, P. Eles, and Z. Peng. A Simulation Methodology for Worst-Case Response Time Estimation of Distributed Real-Time Systems. In *Conference on Design, Automation and Test in Europe (DATE)*, pages 556–561. ACM, 2008.

[53] J. B. Schmitt, F. A. Zdarsky, and I. Martinovic. Improving Performance Bounds in Feed-Forward Networks by Paying Multiplexing Only Once. In *14th GI/ITG Conference Measurement, Modelling and Evaluation of Computer and Communication Systems (MMB)*, pages 13–27. VDE Verlag, March 2008.

[54] C. Schröder, M. von der Beeck, M. Rappl, and P. Braun. Modellbasierte Softwareentwicklung für automobilspezifische Steuergerätenetzwerke. In *Bericht 1646 zur VDI Fachtagung 2001*. VDI, 2001.

[55] M. Schulze, T. Maekinen, J. Irion, M. Flament, and T. Kessel. Final Report of Preventive and Active Safety Applications Integrated Project. 2008. http://www.prevent-ip.org.

[56] SimTD Consortium. SimTD Project Report on Wireless Car2X Communication in Road Traffic. 2008. http://www.simtd.org.

[57] L. Thiele, S. Chakraborty, and M. Naedele. Real-Time Calculus for Scheduling Hard Real-Time Systems. In *IEEE International Symposium on Circuits and Systems*, pages 101–104. IEEE, May 2000.

[58] K. Tindell and A. Burns. Guaranteed Message Latencies for Distributed Safety Critical Hard Real-Time Networks. Technical Report YCS 229, Department of Computer Science, University of York, UK, 1994.

[59] University of Uppsala, Sweden and University of Aalborg, Denmark. UPPAAL - An integrated Tool Environment for Modeling, Validation and Verification of Real-Time Systems. http://www.uppaal.com.

[60] H.-C. v. d. Wense, editor. *LIN Specification Package*. LIN Consortium, 2003.

[61] Vector Informatik. CANoe. http://www.vector.com.

[62] Vector Informatik. CAPL. http://www.vector.com.

[63] E. Wandeler and L. Thiele. Real-time Interfaces for Interface-based Design of Real-Time Systems with Fixed Priority Scheduling. In *5th ACM International Conference on Embedded Software (EMSOFT)*, pages 80–89. ACM Press, 2005.

[64] XJ Technologies Company. AnyLogic. http://www.xjtek.com.

[65] W. Zimmermann and R. Schmidgall. *Bussysteme in der Fahrzeugtechnik*. Vieweg + Teubner, Wiesbaden, Germany, third edition, 2008.