Self-organization and Optimization of Priority-based Communication Buses

Der Technischen Fakultät
der Friedrich-Alexander-Universität
Erlangen-Nürnberg
zur
Erlangung des Doktorgrades Dr.-Ing.

vorgelegt von

Ziermann Tobias

aus Ochsenfurt

Als Dissertation genehmigt
von der Technischen Fakultät
der Friedrich-Alexander-Universität Erlangen-Nürnberg
Tag der mündlichen Prüfung: Date of Defense

Vorsitzender des Promotionsorgans: Prüfungsvorsitzender

Gutachter: Professer Dr.-Ing. Jürgen Teich
Professor Zoran Salcic

Abstract

With the increasing complexity in distributed embedded systems, the communication between the nodes becomes an essential part. A popular approach to realize that communication is to use priority-based communication buses. Due to cost and physical constraints, however, the amount of communication is limited.

This thesis pushes the limitations of priority-based communication buses by two methods: Access control and data rate increase. The access control targets two weaknesses of priority-based communication buses. First, it does not allow fair distribution of bandwidth for equally important message streams. Second, the response time for messages with low priority increase with high bus utilization. Based on a game theoretic model, a reinforcement learning algorithm is proposed that uses simple local rules to establish fair bandwidth sharing. Furthermore, a dynamic offset adaptation algorithm for scheduling messages is introduced that reduces the average response times. This in turn allows to increase the utilization while maintaining the same response times. Both algorithms are executed fully distributed during run-time, require little computational effort and no additional communication. An increase of data rate on protocol level is proposed by extending the Controller Area Network (CAN). By sending additional data in time slots, where CAN-conform nodes do not listen, devices not dedicated to these boosted data rates can still be used. Prototype implementations of all introduced methods demonstrate the feasibility and gains on a real physical setup.

Acknowledgments

I would like to express my sincere gratitude to Prof. Dr.-Ing. Jürgen Teich for his supervision.

This work was supported in part by the German Research Foundation (DFG) under contract TE 163/15-1.

Contents

1 Introduction

Since the beginning of mankind it has been a major goal to *improve*. This was achieved for example by using better tools and optimizing working processes. Probably the earliest and most prominent example is the change of working material from stone to bronze. The current level of capabilities is mainly enabled by two more recent developments. One important step that improved the productivity was made during the industrial revolution. There, development did not enhance the tools that served a human to build things, instead machines were developed that were able to build things by themselves. The second invention that gave a huge improvement was the invention of electricity and the transistor. It provided a huge step in development and opened up many new possibilities for optimization. What started with giant calculating machines advanced to tiny energy efficient processing devices.

Besides general purpose computers the drive for optimization created an additional type of device:

> "... the embedded computer, a computer that is integrated into another system for the purposes of control and/or monitoring." [Cat05]

Due to their specialized nature and advances in transistor miniaturization, embedded systems have become very small in several aspects: Overall size, power consumption, and price. For these reasons, today, these processing devices can be found almost anywhere in our daily life. The complexity of these devices and the interaction between them has reached a point where it becomes impossible for humans to control the behavior directly. In our opinion, a change comparable to the one during the industrial revolution is necessary in order to continue

the progress: The systems need to organize themselves. For systems with *self-organization* property it should be possible that the designer simply specifies goals to the system without needing to specify how they should be achieved. For example, instead of controlling when a traffic signal light should turn green or red, only the goal of minimizing the waiting time should be sufficient.

Rather than trying to achieve this goal generally, in this thesis the self-organization capabilities in the specialized domain of priority-based buses in Distributed Embedded Systems (DES) are investigated.

1.1 Distributed Embedded Systems

Distributed Embedded Systems (DES) consist of many embedded computing nodes interconnected via networks. DES are expected to be the mainstream of future distributed control systems. Connecting multiple computing nodes offers new opportunities and application areas for embedded systems, but also creates new challenges that arise with the increased complexity. The goal of this work is to optimize those systems and to tackle the problem of increased complexity by self-organization.

Examples of such systems can be found in many application domains such as industrial automation, transportation, automotive systems, health care, agriculture, and entertainment. In the following, two examples of such DES are presented. The first example is a printing automation line from Heidelberger Druckmaschinen AG, shown in Figure 1.1. Several steps of the printing process are controlled by switching cabinets. They constantly communicate to ensure the correct function of the whole system. With the never ending drive for improvement in quality and production speed, and decreased costs for electric sensors and actuators, the size and the complexity of these automation lines increases.

The second example is a typical car currently available on the market. Figure 1.2 shows the usual computing nodes, in this case called Electronic Control Units (ECUs), that are available in it. The connection and information exchange between these units enables to decrease cost because of reuse of already gathered information and allows to integrate new applications such as, for example, the electronic stability control (ESC), where all four wheels are braked individually. In the automotive domain, complexity has already reached a point where handcrafted solutions are not feasible. It is impossible to overview the more than 70 ECUs in a modern car [NSL09]. This problem and the need for optimization will increase with the introduction of new types of cars that do not use solely combustion engines as driving mechanism. These hybrid cars using electric motors additionally have further components that need to be controlled. Existing solutions to handle these complex systems will be highlighted in Section 1.2.

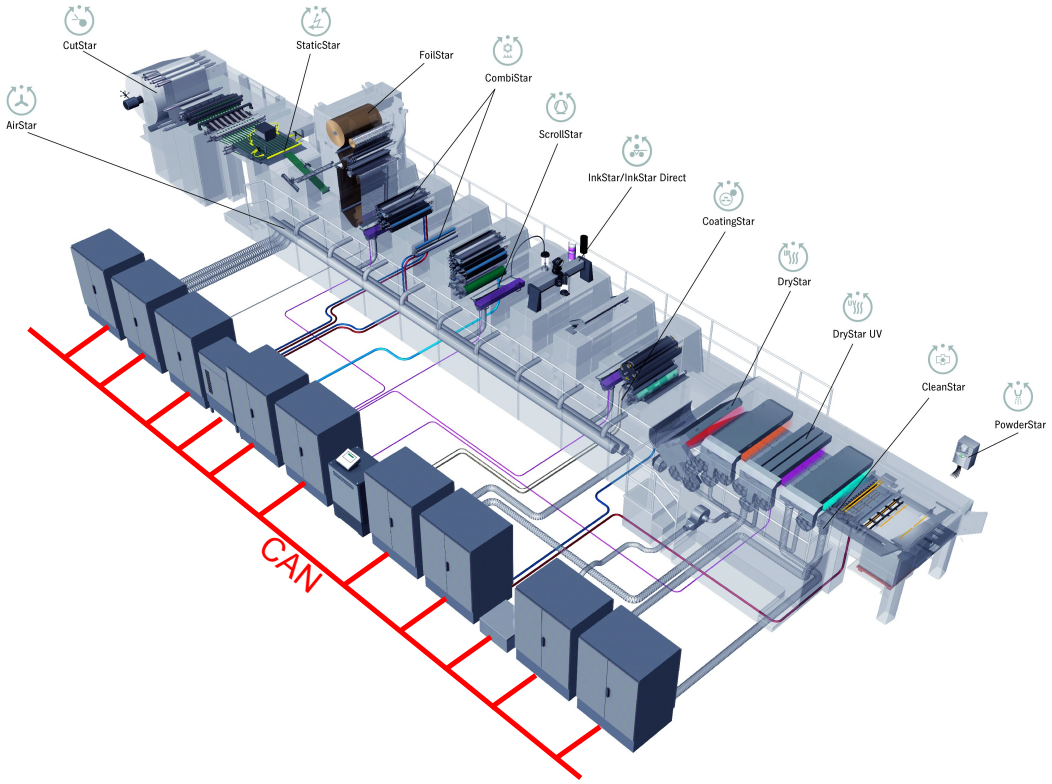


Figure 1.1: Printing automation line [Hei], where the different control nodes are connected by a communication bus to ensure correct interaction of the individual processing steps.

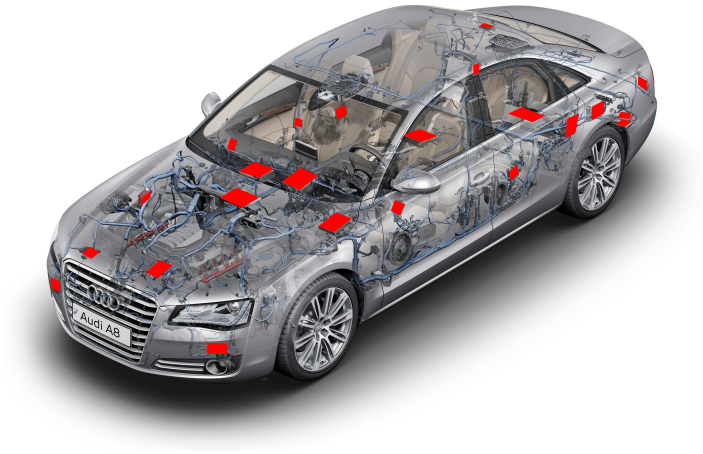


Figure 1.2: Transparent view of a modern car [Aud] with visible ECUs highlighted in red.

The DES we are targeting is characterized by the following properties: Multiple ECUs are connected using a shared communication medium, i.e., a bus. The ECUs are computing nodes that can sense physical signals from the environment and actuate to it. The nodes run applications that use these sensors and actuators and execute control algorithms. Typically, these applications are sensor-controller-actuator chains, where the sensors and actuators are physically distributed and therefore on different ECUs. In Figure 1.3, an abstract view of such a system is shown.

One major challenge is to establish the communication between the ECUs, because the communication infrastructure can be a major bottleneck of the system. This is due to the fact that high speed general purpose networks are not applicable for this type of applications because of safety issues, lack of guarantees that the messages will be transferred in time over the communication infrastructure, cost, and reliability requirements. Instead, special protocols are employed that are tuned to the application requirements. However, the data rate of these protocols is limited, which limits the number of ECUs that can be connected and used in a system. Therefore, the utilization of the communication bus needs to be carefully optimized.

1.2 Communication design for DES

The systems considered in this work appear with different names in the related literature. Different terms are chosen depending on which aspect the corresponding work concentrates on. Very often the attribute real-time is included because the timely behavior is one of the key challenges in DES [Kop11, PC07]. Also the

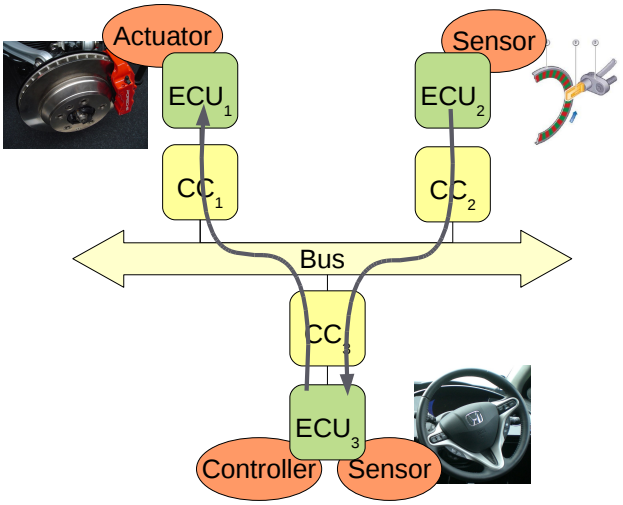


Figure 1.3: Schematic view of a controller-sensor-actuator chain.

term *networked embedded systems* is used to emphasize that the interconnection between the embedded devices is the important characteristic [Str07]. We will use the term DES throughout this thesis to highlight the manifold aspects.

The design of the communication for DES can be classified into two principle approaches. Either the communication is planned *statically*, i.e., trying to consider all possible situations at design time, or the communication system is equipped with tools that *dynamically* adapt to the current situation at run time. Even though a clear division is sometimes impossible, in the following we will present the state of the art of both categories.

1.2.1 Static Approaches

In this section, we will first introduce existing holistic approaches for designing DES and then refining it to communication design where the major contributions of this work are placed. With the growing complexity of electronic systems the wish for developing them at higher levels of abstraction emerged to increase the design productivity. The effort of making this possible is often summarized under the term *system level design* [SV07]. A behavioral specification of the whole system is used to make initial design decisions. In the three so-called *mapping* tasks [Tei12] the system is refined:

1. *Allocation*: Selection of possible computing and communication resources,
2. *Binding*: Mapping of functions to the resources, and
3. *Scheduling*: Choosing when the functions are executed.

The goal of design automation tools, for example [KSF+09], are to automate these mapping tasks allowing the exploration of different design space options. In general, these tools can cope with computation and communication elements. This thesis concentrates on communication elements. Therefore, computation elements are not further explained. Communication elements are not only hardware components, but also software components such as for example different scheduling or different communication protocols. Some software is bound to certain hardware components, such as for example a certain communication protocol only works with the associated transceiver. But with increasing computation power, the current trend is to develop hardware components that offer a high flexibility in the possible communication options. In this thesis, we will mainly focus on the development and optimization of the software part of the communication elements. In particular, we improve the performance of already existing protocols and systems to avoid costs of large changes. Additionally, a new protocol is presented that needs only hardware changes at certain communication nodes.

Another important aspect of design automation is the evaluation of the mappings [PEPP04]. Even though this is not the focus of this work, we will provide options for evaluation at the corresponding places. More existing work of static approaches of the particular communication elements will be presented at the beginning of the corresponding chapters.

1.2.2 Dynamic Approaches

By dynamic approaches we mean in general systems that are able to react to changes during run-time of the system. An analogy exists in the field of control theory [Lun05], where static approaches refer to open-loop controllers and dynamic approaches refer to closed-loop controllers. Static approaches require that the tasks and the environment is completely known at design time (or at least complete knowledge is assumed). On the other hand, dynamic approaches are able to cope to some extend with unknown environments. They could be small building bricks of the system and integrated into a static design flow. For example, a priority-based scheduling allows to cope with changing communication requests. It can be integrated by using analytical or simulative methods to evaluate such a design decision. In the following, we will present the existing work that follows the approach that dynamic approaches should not only be small pieces within the system design, but instead, the design method itself needs to be dynamic.

The first idea for this change of thinking was the project proposed by IBM called *Autonomic Computing* [KC03]. This research area mainly focuses on managing the increased complexity in server architectures by creating systems that have the so called self-management property. It means the details of the complex tasks of the system operation and maintenance should be dealt by the system itself without human interaction. Systems with these properties should be able to detect and correct errors on their own, use its components optimally, configure these components and appraise possible attacks and fend them off.

The idea of Autonomic Computing was followed and extended by the research field *Organic Computing* [Sch05]. It goes beyond server architectures and addresses more general systems that lead to a global self-organizing behavior using only local rules and therefore appear to be "life like" (organic). Additionally, a major difference to Autonomic Computing is the goal of being able to control the global self-organizing behavior often referred to as *emergence* [MMS06]. The behavior of self-organizing systems often show very good properties regarding scalability and robustness against disturbances and parameter changes. For this reason, these systems are particular suitable for future complex technical systems.

While several projects do research on these issues in a general way [Ger05, DDNDM07, SBP+09], many other projects build self-organizing systems for

special application domains. Especially within the Priority Program "Organic Computing" of the German Research Foundation [DFG11] many problem specific solutions in very different fields were developed. The work presented in this thesis belongs to the group of specialized organic approaches. In our opinion, OC is more a paradigm or a way of thinking than a specific methodology. Nevertheless, there are some general design patterns like the observer/controller architecture [RMB+06] that are used and help in creating systems at all layers of abstraction from low-level specific problems [LHR+05] to high-level system design [See11].

Besides the way of creating systems, Organic Computing also describes the properties of a system:

> "OC [Organic Computing] defines an organic computer as a self-organised system that can adapt to a dynamically changing context and achieves the so called selfx-properties ..." [RMB+06]

While the terms selfx/self-* or self-organizing are used very often in the technical context, in the following paragraph, we try to overview and clarify how this term is used in this work. The first part of the terms "self" means that the system on its own achieves a certain property without or very few external control. The x or * stands for the property the system should have [SMSc+10]. For example, the Autonomic Computing Initiative had as goal that systems should be self-configuring, self-healing, self-optimizing, and self-protecting. However, it could be any property. The term "organizing" takes a special position within these properties: A system that has the self-organizing property needs to be able to be split into subsystems. These subsystems have their own local behavior that achieves a global goal, in other words, the subsystems organize themselves. The global goal can be to maintain constant properties under changing environment or system setup, or improving properties compared to a static approach.

One of the main challenges is to find these local rules that together result in a self-organizing behavior. Additionally, it is even more important that this global self-organizing behavior corresponds to the desired goal. The design and evaluation of such systems in the context of bus-based communication is the main focus of this work.

The following quotation shows some difficulty that is associated with the use of "self" for technical systems.

> "... the well-known fact that the meaning of self is dependent on where we draw the system border." [RMB+06]

To avoid this uncertainty, we hereby define the system border around a single DES. Even though, drawing the border around a single embedded system or around several DES is possible and is certainly of interest for research, it is not part of this work.

We will shortly highlight by an example how self-organization is used in communication systems already today. In the domain of computer networks self-organization is one of the key concepts. For example, the congestion control of the Transport Control Protocol (TCP) [Flo01] implements a decentralized mechanism to handle congestion by using a local control rule which increases the transmission rate when packets arrive correctly and reduces it upon packet losses. Thus, there is no explicit management of network resources, but they are shared among all participants in a fair manner. This avoids congestion in any case which would lead to a degradation of communication quality for all participants.

1.3 Contributions and Collaborations

As stated throughout the introduction, the focus of this work is on communication of DES. In particular, priority-based communication buses are improved by self-organization or modification of the protocol. This allows flexibility in adding and removing nodes at run-time. Furthermore, an increase of communication can be tolerated without changing the existing communication structure.

The contributions are divided into three main parts corresponding to Chapter 3, 4 and 5:

1. decentralized fair bandwidth assignment,
2. dynamic adaptation of offsets, and
3. increasing the data rate with CAN+.

Parts of the solutions and results presented in this thesis were previously published. A list of all publications authored and co-authored by the author of this thesis may be found starting at page (TODO: make own bib(siehe Kommentare am Ende). In this section, all publications by the author leading to and culminating in this thesis are summarized. A publication presenting parts of the results in Chapter 3 and 4 summarizes the work on the project Organic Bus [ZWT11*].

1.3.1 Decentralized Fair Bandwidth Assignment

Priority-based access on communication buses makes it difficult to provide equal bandwidth to message streams with equal importance. The difficulty is that caused by the protocol each stream has its own priority. This leads to a preference of the stream with the highest priority. In order to analyze this problem, Stefan Wildermann developed the priority-based medium access game [WZT09*]. He showed that with an adaptation of the utility function an equal

bandwidth sharing is a stable state. Based on this game theoretic model, we developed reinforcement learning algorithms that are able to reach this stable state [ZMWT10*]. Extensive simulations show the performance with different parameter settings. Furthermore, it is proven that the algorithm based on probabilistic access enables equal bandwidth sharing even in the presence of further higher priority traffic [ZWMT11*]. For the algorithm based on periodic access, a procedure to set the learning parameter is developed. Besides round-based simulations for the evaluation of the theoretical game, bit-accurate simulations of the bus protocol Controller Area Network (CAN) is used to show the results for a more realistic setting. Finally, using the developed prototype system, a fair sharing of bandwidth for up to four video streams can be demonstrated.

1.3.2 Dynamic Adaptation of Offsets

Real-time applications that are periodically executed on the priority-based communication bus require short message response times. Due to growing system and functional requirements, low capacity of the communication bus and usually strict conditions under which it is used, it gets more and more difficult to achieve short response times. An approach for achieving high utilization is presented by proposing a Dynamic Offset Adaptation Algorithm (DynOAA) for scheduling messages and improving message response times without any changes to the communication protocol [ZST11a*]. This simple algorithm, which runs on all nodes of the system, results in excellent average response times at all loads and makes the approach particularly attractive for soft real-time systems. We first address single bus (segment) systems and then extend the model and the algorithm to systems that use multiple buses (segments) connected by a communication gateway [ZST11b*]. Experiments show (1) the robustness of the algorithm in the case of fully asynchronous systems, (2) the ability to deal with systems that change their configuration (adding or removing message release nodes) dynamically, and (3) the performance in systems containing multiple bus segments connected by a gateway. In addition, we propose a method for self-initialization of the communication system on its start-up, which reduces response times from the very start of the system [ZST12*]. We demonstrate the performance improvements by comparing the approach with other approaches qualitatively and quantitatively. Furthermore, a hardware testbed is developed that allows the measurement of the message response times using the algorithm on a real DES using FPGAs [ZBZT12*]. Using this testbed, we develop a pure hardware and a pure software implementation. They show that both implementations are possible. However, parts of the software implementation require a significant amount of computation. As a result a mixed HW/SW implementation is proposed [ZST13*].

1.3.3 Increasing the Data Rate with CAN+

As the number of electronic components in automobiles steadily increases, the demand for higher communication bandwidth also rises dramatically. Instead of installing new wiring harnesses and new bus structures, it would be useful, if already available structures could be used, but driven at higher data rates. Therefore, we propose an extension of the well-known protocol CAN, called CAN+ with which the data rate can be increased without interfering with the slower standard communication [ZWT09a*, Zie08*]. An implementation of CAN+ on an FPGA is provided to prove the feasibility and the impressive throughput gains of 1 Mbyte/s, that is 16 times the standard rate [ZWT09b*]. In order to show that electromagnetic compatibility is similar compared to standard CAN, the developed prototype is measured [ZT10b*]. It is shown that CAN+ can be used in a video application with dynamic partial reconfiguration of FPGAs [ZT10a*].

1.4 Outline

The remainder of this thesis is organized as follows. Chapter 2 lays the fundamentals for the remainder of this thesis. First, we define a model of bus-based DES communication based on message streams. Next, priority-based protocols are presented. Here, the main focus lies on CAN because it is used as example throughout this thesis. Chapter 3 proposes a self-organizing mechanism to share bandwidth equally on a priority-based medium. Chapter 4 explains how response times can be minimized using distributed online adaptation. In Chapter 5, a new protocol is introduced, that makes it possible to communicate with different data rates on an existing CAN-Bus without disturbing the original CAN conform communication. Chapter 6 presents a prototype DES based on FPGAs to test the preceding methods and algorithms. Finally, conclusions and future work are given in Chapter 7.

2

Priority-based Single-Hop Communication Systems

Basically, communication systems can be divided into *network-based* and *bus-based* systems. In the latter one, each node can communicate with other nodes directly by being attached to bus lines which are shared between all communicating nodes. Whereas in network-based systems, communication between nodes is only possible by routing messages over one or more other nodes. The main challenge in network-based systems is the routing problem, i.e., determining a path through the network from the sender to the receiver of a message.

Although network-based communication offers a good scalability so that huge heterogeneous networks can be built, e.g., the Internet, it is difficult to predict timings, and to analyze and guarantee real-time requirements. In contrast, bus-based communication systems, as presented in this chapter, often scale poorly and may result in performance bottlenecks. However, they provide lowest cost solutions and are easier to analyze and predict. Consequently, they are more commonly used in DES. Therefore, in the following, we will only consider bus-based communication.

Having several communication nodes connected to one shared medium raises the problem of arbitration. If more than one node is sending concurrently then the transmitted information will be disturbed and no information can be communicated. There exist several mechanisms to solve the multiple access problem. One solution is using different channel access methods such as for example different frequencies or different codes. A technically much easier solution, however,

is to ensure that only one node is sending at a time. Here, again, two methods can be differentiated: *event-triggered* and *time-triggered access*.

Event-triggered means that the transmission of a communication message is triggered by an event that can occur randomly. If the nodes transmit messages at any point in time still collisions occur. This is solved by listening to the bus before transmission and only send if there is no ongoing transmission. This method is very often referred to as carrier sense multiple access (CSMA) [KT75]. However, due to line delays it is still possible that two or more nodes start transmission at the same time. In this case, the collision can either be detected and the message retransmitted at a random later point in time (collision detection) or the method of bit-wise arbitration can solve the conflict by granting access according to priorities (bit-wise arbitration). In the following, we will refer to the bit-wise arbitration as *event-triggered priority-based mechanism* to emphasize the properties. How it works in detail is explained in Section 2.2.2.2.

Time-triggered means that the transmission of communication messages is triggered according to a fixed time schedule. The at design time arranged schedule ensures collision free communication. A typical representative is the time-triggered protocol (TTP) [KG93], but also Flexray [FC09] offers this mechanism. A comparison of time-triggered and event triggered approaches has been subject of many discussions [Alb04], and there is a sound argument of the advantage of using time-triggered approach for systems with purely periodic message release. Nevertheless, this work focuses on event-triggered priority-based communication, because it offers great flexibility in changing the communication order, but also makes it possible to guarantee real-time properties because of the deterministic priorities.

This chapter is organized as follows. In Section 2.1, basic definitions of the applications and their requirements are presented. In Section 2.2, the most popular priority-based bus protocol, CAN, is introduced. It serves as the main target protocol in this thesis and is therefore described in detail. Finally, in Section 2.3 two other priority-based protocols are highlighted.

2.1 Basic Definitions

The system we are targeting can be described by a set of nodes communicating over a shared bus medium, as shown in Figure 2.1. In real applications, most of the nodes or all nodes are connected to the physical world by sensors and actuators. As described in Section 1.2.1, there exist holistic approaches for design and analysis considering the whole DES including applications. However, only approaches based on static assumptions exist that are not self-organizing. In order to allow the development of a self-organizing system level design methodology, first the basis has to be build. In this thesis, we provide a building

brick by focusing on the communication between the nodes. Here, one or more tasks on each node may initiate a communication, i.e., release messages. In our model, we abstract the tasks by considering only the mechanism used to release *messages* called a *stream*.

A stream s_i can be characterized by a tuple (N_i, U_i, R_i), that is, by a node N_i the stream is running on, an occurrence scheme U_i that describes when a message is released by stream s_i, and a requirement R_i of the amount of messages or the time until the generated message should or must be transmitted. A *scenario* consists of n streams. We are assuming a message-oriented non-preemtive priority-based bus access, meaning once a message has started being transmitted it will finish transmission. Additionally, if two or more streams try to transmit concurrently the stream with the highest priority transmits. The priority by which access is granted to the bus is given by the numbering of the streams. In particular, the stream with the lower index, stream s_i, has a higher priority than a stream with a higher index s_{i+1}.

The target or possibly multiple targets of a stream are not considered because the bus structure allows every node to read every message. In consequence, it does not influence the schedule and can therefore be ignored. The workload w describes the utilization of the bus. It is defined by

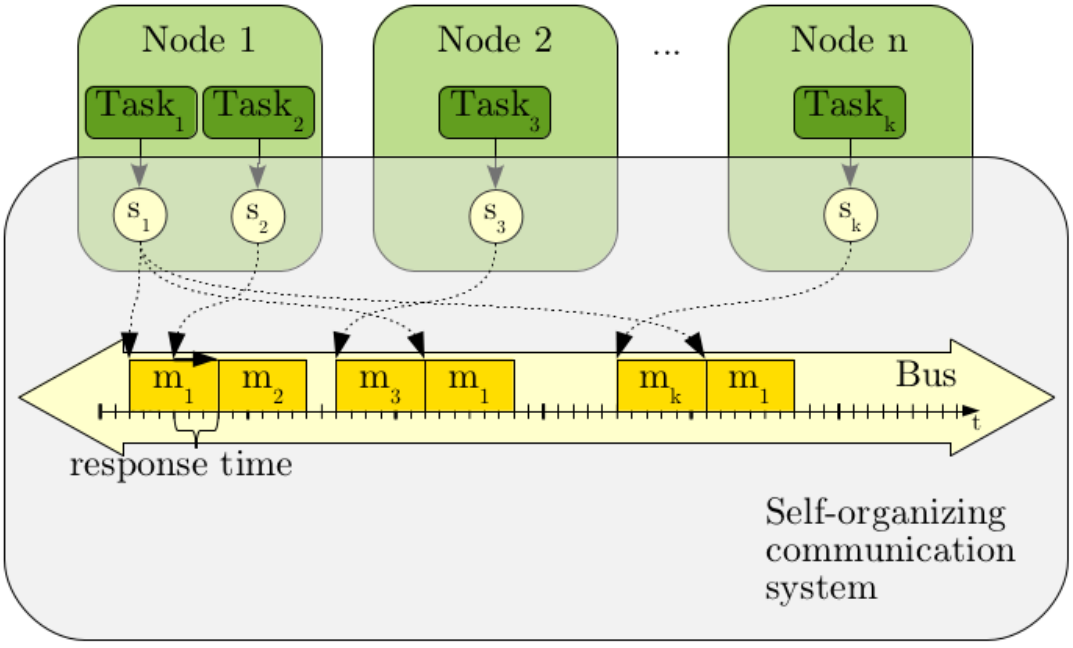


Figure 2.1: System model for priority-based bus communication with y nodes and n streams.

$$w = \frac{t_{busy}}{t_{busy} + t_{idle}},$$

where t_{busy} is the time the bus is busy and t_{idle} is the time the bus is idle.

A *message* m is a single release such as for example a CAN frame of the stream (cf. Section 2.2.2.2). The time between a message release and the start of its uninterrupted transfer over the bus is the *response time* of the message. We omit the constant non-preemptive time to transfer the message to the response time, because thus a response time of zero is always the best possible case for every message independent of its length. This will simplify later comparisons between different techniques to influence the scheduling of messages. In Figure 2.1 for example, the response time of message $m_{2,1}$ is three time slots, because it is delayed by the running message $m_{1,1}$. The *worst case response time* $WCRT_i(t_a, t_b)$ of a stream during a certain time interval starting at time t_a and ending at time t_b is the largest response time of the messages of the stream recorded during that time interval. An analytical approach calculates $WCRT_i(0, \infty)$.

In classical real-time models [SRSB98], the occurrence U of a task, here stream, can have the following properties: *Periodic* or *sporadic*. Periodic messages are released with a fixed rate and are characterized by a period T_i and an offset O_i. The offset is the time until the first message is released relative to an imaginary global time reference. Because the local time reference differs from the global one, the offsets can drift over time. The *hyper period* P is the least common multiple of all periods $lcm\{T_1, T_2, ..., T_k\}$. Assuming a synchronous system, the schedule is finally periodic with the hyper period.

In our analysis, we assume an offset free system as defined in [Goo03]. This means the offsets of individual streams are not bound by any constraints, but rather can freely be set. This approach potentially enables the avoidance of conflicts by setting the offsets appropriately [GGN+06]. Sporadic streams release messages irregularly but with a bounded interarrival period. This allows us to treat sporadic streams in the worst case as periodic streams. Therefore, we only consider periodic streams in the following. In addition to these classical models, we define the occurrence property *probabilistic.* It characterizes the occurrence of a stream as a probability at which a message is generated during one transmission slot.

The requirements R of the streams are now split into two groups: *response time sensitive* and *throughput sensitive*. The former requirement can be defined by a deadline. In the case of a *hard deadline*, the response times of a stream must always be below the deadline. If the requirement is a *soft deadline,* it is desirable that as many response times as possible are below the deadline. However, the requirement for response time sensitive streams can also be that the average or the worst case response times should be as small as possible. For

throughput sensitive streams, it is not important to meet deadlines but it is necessary to send a certain amount of messages per time unit. It is also possible that the quality of the application varies with the bandwidth it can use, so the requirement is to send as many messages as possible. This kind of requirement is called *bandwidth*.

Any combination of occurrence and requirement can be imagined. However, in this thesis, we distinguish between three types of streams explained in the following sections.

2.1.1 Hard Real-time Streams

Safety critical applications are represented by hard real-time streams. These applications will lead to a system malfunction when the deadlines are not kept, so the requirement R is a hard deadline. The occurrence of hard real-time streams is assumed to be strictly periodic, otherwise it would be impossible to guarantee any behavior. The only possibility to guarantee that the deadlines are always kept is to analyze all possible situations the self-organizing communication systems can reach before start of the system.

Still, we have to assign a priority to each stream, which is not changed by any self-organizing mechanisms. If all deadlines are equal to the period, the priorities are static, and the messages can be preempted, it is known that *rate monotonic assignment* is scheduling optimal [Par07]. Scheduling optimal means that if there exists a priority assignment that keeps all deadlines then the rate monotonic assignment keeps all deadlines. It assigns the streams with the smallest period the highest priority. If the deadlines are less than the period, Rate Monotonic is not scheduling optimal. In this case, *Audsley's algorithm* [Aud91] is scheduling optimal. For the non-preemptive case, as considered in this work, Audsley's algorithm is scheduling optimal for the general case [GRS+96]. Even though an optimal solution is desirable, the main concern in system design lies in the verification that all streams transmit their messages within their deadline under the given setting. In the following, we will present several approaches to analyze the response times of the streams.

The simplest and most used approach was first introduced in [THW95] and revised in [DBBL07]. To calculate the worst case response time of each stream, we assume that asynchronous fixed priority non-preemptive scheduling of messages is implemented. The response time then consists of two elements: *blocking time* $B(m_i)$, due to lower priority messages which can't be preempted, and *interference* $I(m_i)$ due to higher priority messages. The upper bound for blocking time $B(m_i)$ is the transmission time of the largest message in the system. The interference $I(m_i)$ can be calculated by the following construction: Start all streams with an offset of zero. The time until the first occurrence of the analyzed stream is the interference. In summary, the Worst-Case Response

Time $WCRT_i(0, \infty) = B(m_i) + I(m_i)$. By comparing the Worst-Case Response Times (WCRTs) to the deadline for each stream, we can decide whether the system can hold all constraints. Klehmet et al. [KHHG08] apply the method of Network Calculus to predict the transmission delays beforehand.

In addition to the specialized approaches, a large variety of holistic approaches for DES exist, for example [CKT03, LPY97]. They take into account the whole system including the timing of tasks running on resources and their communication. Similar to the specialized approaches, the timing analysis is done offline during the design phase and cannot adapt to system changes. A further disadvantage is that the computational complexity is very high, because the holistic approaches consider a lot of constraints.

One of these approaches is the compositional performance analysis methodology used in SymTA/S [HHJ+05]. It solves the global timing analysis by decomposing the system into independently investigated components. The timing between the tasks (event streams) is then captured with event models that can efficiently be described by a small set of parameters. Based on this offline analysis, an online analysis was developed as described in [SE08]. To analyze the system distributed online, models of the application running on each embedded system as well as its architecture are stored on the embedded system itself. The reason for this is that the embedded systems itself analyze the timing properties by communicating timing information between the nodes until the timing properties converge to a stable state. In order to allow online change of the system, a run-time environment is proposed in [NSSE10]. It establishes an abstraction layer between platform and application. Contracts are used to guarantee the real-time constraints of the tasks. These contracts are preverified using formal methods before admitting corresponding configurations to take effect in the system. Thus, the system will only transition between proven safe configurations. Therefore, this approach perfectly suits the requirement for hard real-time streams.

In summary, ensuring that hard deadlines are kept needs to be done before the start of the system. The adaptations done by the proposed self-organizing communication system during run-time do not influence the properties of the analytical calculations. Therefore, they can be used together.

2.1.2 Soft Real-time Streams

The main disadvantage of the methodology used for hard real-time streams is that they are overly pessimistic and very often a big part of the communication capacities remain unused. For many practical systems this unused capacities would require additional hardware, which would be too expensive to integrate. Therefore, the majority of streams in practical DES are soft real-time streams.

The main characteristic of soft real-time is summarized very good by the following quotation.

> "The consequences of not meeting timing constraints in soft real-time systems are not as disastrous as hard real-time systems. Soft real-time systems are still considered functionally correct if timing constraints are not seriously violated" [TBY+96]

An approach to quantify this characteristic is the introduction of $(i, j) - firm$ deadlines [HR95]. An application is assumed to still function correctly, if at least i number of deadlines of j are kept. However, this approach requires the existence of detailed timing constraints, which can generally not be assumed. For example for most control algorithms it is impossible to find a deadline where the algorithm fails.

In this thesis, we do not consider timing constraints for soft real-time streams because the focus of this work is the optimization and not the evaluation. Instead, we assume that it is desirable to reduce the response time to a minimum. Typical examples of such applications are found in the body network of an automotive embedded system [NSL09]. For example, a rain sensitive wiper needs to react in time on the changing environment, but some delayed messages will not cause a system failure. Further details on the evaluation used are provided in Section 4.2.

2.1.3 Bandwidth Streams

In contrast to the first two types of streams, the response time is not the critical requirement for bandwidth streams. Instead the amount of data sent during a certain time interval is crucial. Streaming applications, like audio or video streaming, are represented by bandwidth streams. On the one hand, their quality is depending on the amount of data they can send. For example, a video stream needs to be compressed less when there is more bandwidth available. On the other hand, these streams are very flexible in the amount of throughput they need. For example, using lossy compression the needed throughput can easily be reduced by an order of magnitude. This means, it is not so important to get a certain throughput, rather it is important that all bandwidth streams get the same amount of throughput. The requirement is bandwidth and the occurrence can either be probabilistic or periodic. The occurrence can be freely chosen (within reasonable bounds), so the objective of the organic system is to find the optimal occurrence.

2.2 Controller Area Network

Today, the most common bus for distributed real-time control systems is CAN [CAN91]. In the following, applications and the structure of the protocol are presented.

2.2.1 Applications

CAN is used in many different application domains for DES. As it was created by Bosch in the early 1980s with the target to decrease the amount of wiring in cars, the main application domain still is automotive. For example, more than 70 ECUs are used in a modern personal car [NSL09]. Most of these ECUs are connected by several CAN buses. Many other transportation systems take advantage of the CAN protocol. For example public transportation buses use CAN to communicate between the electrical circuits and ECUs that are essential for driving, braking, suspension, door opening, and security [MPDO09].

Apart from vehicle communication networks, CAN is also used in stationary applications, where a low cost communication with fast response time is needed. One example is the use in lighting control [DRH+08]. The authors propose a networked embedded system consisting of multiple lighting sensors and actuators connected by CAN. The main characteristic why CAN is chosen is the non-master capability, allowing flexible control of individual and groups of lights. Another example is the use in distributed power generation systems [BHR+09]. In [DA10], an energy efficient switchable distribution transformer has been implemented connecting the control modules by CAN. It shows that the use of the bus allows parallel job handling and therefore better control performance. Many more applications can be found on the website of CAN in Automation e.V. [iA].

2.2.2 Protocol

In the following, a short introduction to CAN will be given. The protocol can be divided into to two ISO/OSI layers: physical layer and data link layer.

2.2.2.1 Physical Layer

The purpose of the physical layer is to find a suitable physical and electrical specification. In CAN, bits are represented by the very simple Non-Return-To-Zero method. This means that binary sequences are transmitted by applying the symbols for the binary values directly after each other without any third symbol. In CAN, a two wire bus is chosen where a logical zero is represented by a voltage difference and a logical one by no voltage difference. Synchronization

of the distributed nodes happens on the falling edge. Possibly long sequences of ones or zeros would make it difficult to synchronize, therefore, bit-stuffing is applied, where the sending nodes add an inverted bit after five consecutive equal bits. The receiver reads this additional bit, but does not pass it to the next higher layer. Furthermore, the concept of dominant and recessive bits is used, leading to the behavior that, if two or more nodes simultaneously try to send, then the dominant bit, here a logical zero, is always approved. This mechanism is the basis for the arbitration of concurrently sent messages explained in Section 2.2.2.2.

The wiring is done by twisted pair cable (*Bus High* and *Bus Low*) and set up as shown in Figure 2.2. Typical setups use a base voltage of 2.5 V, so a logical one is represented by applying 2.5 V to both wires. A logical zero is represented by applying 3.5 V to the Bus High wire and 1.5 V to the Bus Low wire. The length of a bit or the *bit time* t_{bit} is the time from beginning of transmission of a single bit to the end. It is determined by the bit rate r used and equal for all nodes. It is computed as follows:

$$t_{bit} = \frac{1}{r}$$

Possibly the resulting bit signal is not stable for the whole duration of one bit transmission for several reasons. First, the transceivers need a certain amount of time to change the voltage depending on the quality of the transceiver. Therefore, when a bit is different from the previous one it takes some time until the new bit is visible on the bus. Furthermore, wires and transceivers are not perfect, so the signal might become distorted, e.g., by electromagnetic interference such as by an electric motor. In addition, propagation delay causes signals from different nodes to be visible at different times. The CAN specification offers a solution by dividing the bit time in four intervals and sampling the value only

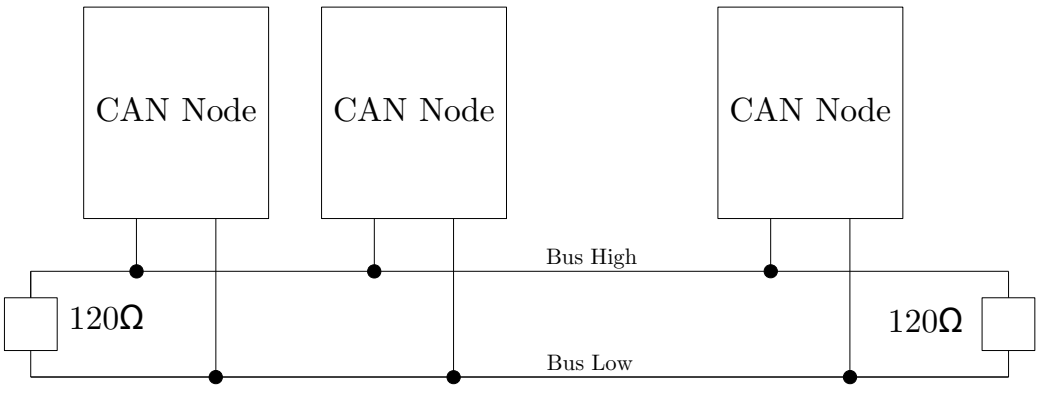


Figure 2.2: Physical example setup of a Controller Area Network

at a stable point, the *sample point*. The length of these intervals is a multiple of the basic time unit called Time Quantum (TQ). Figure 2.3 shows these intervals. The Sync_Seg has the length of one TQ. A possible edge due to a change of the bit value is expected in that interval. The Prop_Seg is one to eight TQ long and is used to compensate the signal propagation delay. The purpose of the Phase_Segs is to correct possible edge phase errors by shortening or lengthening them on demand. The length of Phase_Seg1 is one to eight TQ. The length of the Phase_Seg2 is the maximum of Phase_Seg1 and the information processing time, which is less than or equal to two TQ.

Given these interval specifications, we can derive two extreme cases for the relative position of the sample point. First, assuming the length of the Prob_Seg is at maximum and the length of the Phase_Segs at minimum. This results in the following relative position of the sampling point t_{sample}:

$$t_{sample} = \frac{1+8+1}{1+8+1+1} = \frac{10}{11}$$

The second extreme case occurs when the the length of the Prob_Seg is at minimum and the length of the Phase_Segs at maximum:

$$t_{sample} = \frac{1+1+8}{1+1+8+8} = \frac{10}{18} = \frac{5}{9}$$

In real practical systems the sample point lies at around 2/3 of the bit time. Due to improvements in the accuracy of the transceiver and controller chips, the trend is to move the sampling point further to the end. This allows to tolerate longer propagation delays and therefore longer bus wires at the same data rate. The reason why is further explained in Section 5.1.

2.2.2.2 Data Link Layer

For the data link layer a message-oriented approach is chosen. The CAN specification offers four message types. The most important is the data frame, used for data exchange. Its structure is shown in Figure 2.4. Error handling is done by using a dedicated error message. It is sent any time by any node noticing an error, e.g., stuffing error (after five equal bits the stuffbit is missing) or cyclic

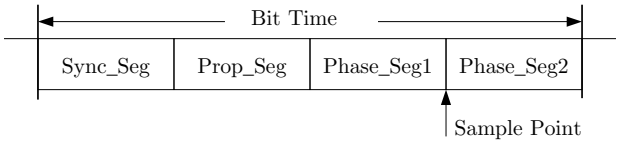


Figure 2.3: Partitions of the bit time of one CAN bit.

redundancy code error. The error message is represented by a sequence of six consecutive dominant bits (logical zeros), which in standard communication is precluded due to bit-stuffing as described in Section 1.3.1. This representation makes it possible to distinguish the error message from any other transmission and is therefore always detected. The specification provides three error states for each node to cope with erroneous nodes. The states reached are depending on the amount of error messages counted and can lead to a turning off of the defect node.

The most characteristic feature of CAN is the priority-based arbitration mechanism. Because nodes start to transmit as soon as data is available to send, it is possible that more than one message starts transmitting at the same time. The method of bit-wise arbitration is used to avoid collisions. It can be described as follows: Each message starts with an identifier sequence and each node transmits and samples in parallel. As soon as a node is sampling a bit different from the one sent by it, it stops transmission. Since zero is the dominant bit, it is assured that the message with the lowest identifier remains at the end of the arbitration phase and transmits its data. This implements a priority-based access scheme where messages with lower identifiers have higher priorities for accessing the bus. The example in Figure 2.5 displays the principle. There, three nodes

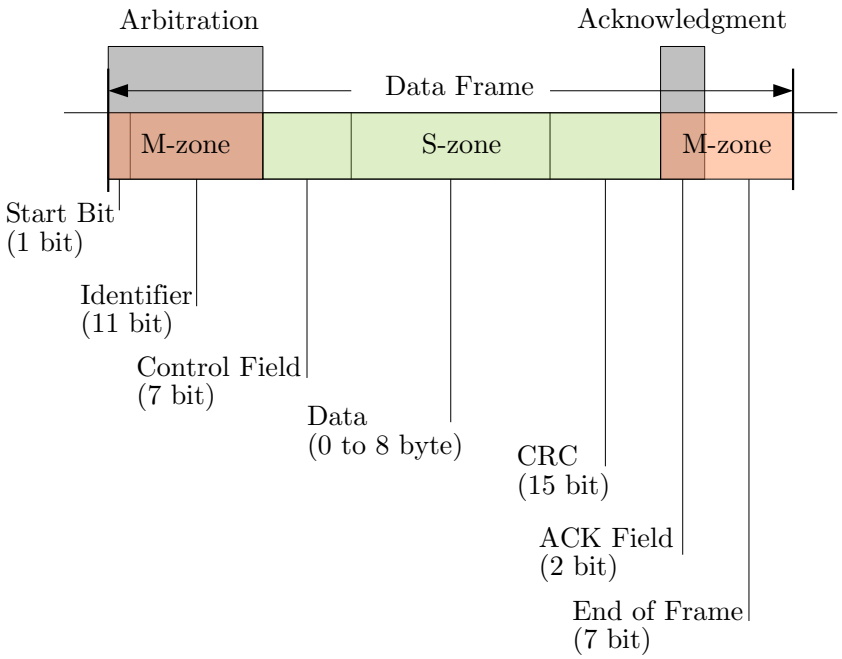


Figure 2.4: The CAN data frame.

start transmitting concurrently. Node 3 has the lowest identifier, node 2 the highest, and the identifier of node 1 is in between the others.

The arbitration mechanism demands that each identifier is uniquely use by one node. Otherwise, two nodes with the same identifier would win the arbitration concurrently and any difference in the control or data field would lead to a collision resulting in an error. Nevertheless, one node can use several identifiers. This is very often used to assign each application or even each variable of an application an identifier. However, the number of identifiers is limited to 2032, because of the fixed length of the identifier of 11 bits and the restriction that the most seven significant bits must not be all recessive.

2.2.3 Properties and Limitations

We divide the main properties that characterize the CAN protocol into the following three categories: Response time, data rate, and error handling. As described in the previous section, in systems that use the CAN bus, a priority is assigned to each message, and, when requesting the bus access, the message with highest priority is granted exclusive access, while the other messages have to wait and retry transmission. This ensures deterministic behavior and short response times for messages with high priority but also potentially big latency for messages with low priority, because the access can be denied more often. Algorithms to overcome this problem are presented in Chapter 4. Additionally, the prioritization leads to problems when equal priority should be assigned to several computing nodes. In Chapter 3, this topic is covered. The arbitration and acknowledgment mechanism is also the cause for the limitation of the data

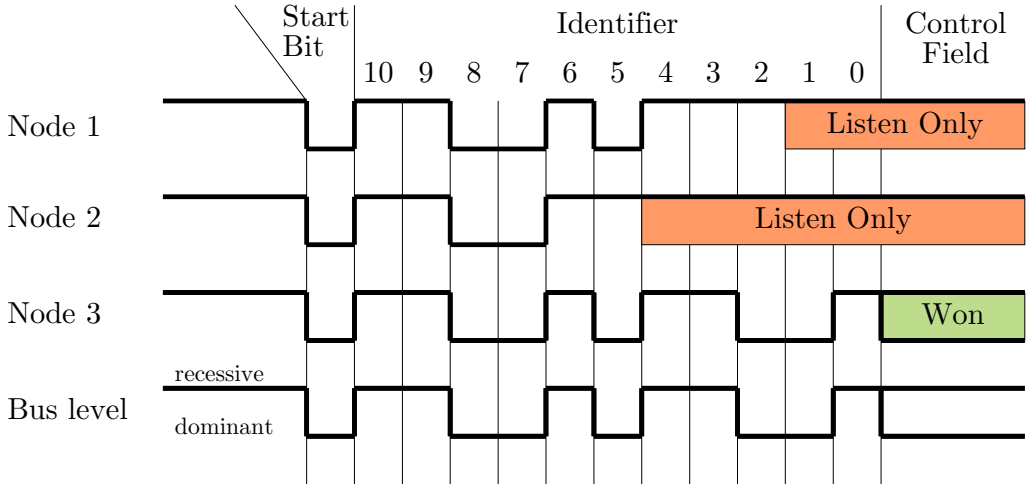


Figure 2.5: Example of the arbitration procedure of CAN.

rate depending on the length of the bus. More details on the problem and a solution are provided in Chapter 5. Even though error handling is not considered in the remainder of this thesis, we will highlight some mechanisms and research results on this topic to show the usability of CAN in future DES.

As already mentioned in Section 2.2.2.2 the CAN protocol itself provides elaborate error detection and self-checking mechanisms [Cha94]. Namely they are transmitter-based-monitoring, bit stuffing, Cyclic Redundancy Check (CRC), and message frame check. This ensures that temporary disturbances, for example from electromagnetic radiation, are detected and a retransmission is initiated. In [ATDP10], a scheduling of the messages using to some extend dynamic priorities is proposed that ensures transmission of critical messages within so called feasibility windows. This scheduling allows to guaranteed the desired levels of fault-tolerance at the price of using more than one priority per message. Additionally, the error detection mechanism of the CAN protocol itself detects certain permanent errors by counting the number of erroneous messages and disabling of faulty nodes. In [SSA06] several possible enhancements are proposed to solve some remaining known issues. For example, a permanent link failure anywhere on the bus line leads to a total falling out. This can be avoided by using a star topology at the expense of additional wiring costs. The *babbling idiot* problem where one node permanently overwrites the communication could be avoided by using an additional bus guardian to control the access.

In summary, the CAN protocol has its limitations and partially there are solutions to it. But even with the limitations, CAN has served as the leading protocol for low cost, short response time communication and we agree with the authors of the following statement on the further importance of CAN in the future:

> "Although it is too early to identify the best automotive networking solution at the backbone level, we believe that CAN is currently the best networking technology at the subnetwork level." [PPA+09]

2.3 Other Protocols

Even though CAN will be used as main case study, in the following two other protocols will be presented that support event-triggered priority-based arbitration. The main characteristics will be described and the differences and commonalities in comparison to CAN are highlighted.

2.3.1 Flexray

A trend towards adding more nodes to the distributed embedded and control systems applications faces limited bandwidth and long message response times

as the major obstacles. An obvious solution to this problem is to try using a faster protocol such as FlexRay [FC09]. However, to achieve a faster communication several aspects have to beconsidered. One aspect is the improved bit rate. The used bit encoding scheme is Non-Return-To-Zero similar to CAN, see Section 2.2.2.1. However, due to a beforehand defined schedule that ensures that timing errors are avoided it is possible to use a bit rate of 10 Mbit/s. Another important requirement for the high bit rate is the synchronization of the nodes. Therefore, a complex synchronization mechanism using different hierarchical time units that are adapted continuously is specified. Going hand in hand with the synchronization is a well defined start up mechanism that enforces the nodes to listen and synchronize to the bus before being able to transmit.

In addition, a different arbitration method is used. FlexRay uses two access mechanisms. First, a priority-based event-triggered access in a-priory defined order (*dynamic segment*) is possible. Second, a static time-triggered access is offered (*static segment*), where the schedules are fixed at planning time. The dynamic segment is scheduled similar to token passing, sometimes also called flexible TDMA: An order of the different applications is specified offline. Following this order, each application can either transmit or pass the token. At the beginning of the next dynamic segment it starts again with the first one. This approach has larger response times than CAN because (1) the messages have to wait for the dynamic segment to start, (2) for the token to be passed, and (3) possibly even to wait for several segments, because the token did not reach the application before the segment is completed.

The static segment has the typical properties of a time-triggered access mechanism: Periodic time slots are reserved statically at design time. This has the advantage that the response times for ideal periodic applications are deterministic, given that the application triggering the message is synchronized with the communication. But for applications that are not active for the full system operation time the disadvantage is that the bandwidth is wasted when the application does not require the time slot. Additionally, the time-triggered access mechanism cannot easily accommodate the system changes during its development and it always has to be planned with complete system knowledge, which is not always available and possible.

Besides the arbitration mechanism and the increased bit rate, the FlexRay [FC09] specification has several differences to CAN. The topology of FlexRay can be a simple passive bus similar to CAN, but additionally an active star can be used to connect the nodes. In this case, all nodes are connected to a dedicated hardware unit that forwards the signals. Both approaches can be combined and a second wire can be used to improve the fault tolerance. Another fault tolerance mechanism is the so called bus guardian that is a separate hardware circuit added to each node to monitor and, in the case of an error, disable the FlexRay communication controller. In contrast to the CAN specification, the

FlexRay specification describes the host to communication controller interface in detail. It exactly defines which parameters need to be able to be configured and how messages are transmitted.

2.3.2 Widom

With the advances in microelectronics the use of wireless communication as a replacement for wired communication becomes technically feasible. The use of wireless communication allows us to save wiring costs. In the automotive domain for example this could reduce the cost and weight of the communication cables itself [ESAT06]. Furthermore, additional savings could be achieved by avoiding difficult constructions at moving parts such as for example doors.

Due to the freedom in spatial layout combined with miniaturization and low power devices, the research area Wireless Sensor Networks (WSNs) [AV10] has evolved. The main idea of WSNs is to use many simple nodes connected by wireless communication to achieve a collaborative effort. Typical applications consist of sensing physical phenomena of large areas over a relative large amount of time. The target applications in this thesis differ from WSNs in terms of real-time capabilities and spatial extend.

Nevertheless, the wireless dominance protocol (Widom) [PAT07] introduced in the context of WSNs fullfils the requirements of event-triggered priority-based communication. The basic idea is to mimic the CAN protocol in the wireless domain. However, it has to be adapted, because the physical properties of the wireless channel are very different from the wired channel. For example, it is not possible to transmit and receive concurrently. The main layout of the protocol will be summarized in the following.

Similar to CAN, Widom uses the concept of messages. The transmission of a message is divided into three phases: synchronization, tournament, and receive/transmit. During the synchronization phase, the nodes receive for a fixed amount of time. If nothing is received during that interval a carrier signal pulse is transmitted to start the tournament phase. During the tournament phase, the identifier of the node is transmitted by sending a carrier signal pulse for a dominant bit and sensing a carrier signal for a recessive bit. If a sensing node receives a carrier signal it knows that a higher priority message is trying to transmit and it changes to reception phase. Similar to the arbitration in the CAN protocol, only the message with the lowest identifier/highest priority is left. This node then transmits its data with the maximal possible data rate of the transceiver.

In contrast to CAN, Widom is not standardized and still in the development. Therefore, many parameters are not fixed yet and a fair performance comparison is impossible. Nevertheless, first prototypes can give estimates for one implementation [Per10]. A custom add-on board to the MICAz platform [Cro]

to support the protocol was developed. It contains three radio frequency chips where two work at 418 MHz and one works at 315 MHz with a maximal data rate of 115.2 kbit/s. More than one radio frequency chip is used to allow faster switching between receiving and transmitting, and to improve time accuracy during synchronization and tournament phase. With this setup it is possible to complete a tournament phase with 10 priority bits in 1754μs. This is still a very high overhead compared to CAN, where the arbitration with 11 priority bits including the synchronization with a data rate of 125 kbit/s takes only 96μs.

Even thought the data throughput of Widom is much lower than CAN it is possible to achieve the same properties of medium access, if the length of a message is limited to a fixed maximal size. This means that all methods in this thesis used with CAN can be directly applied to Widom.

3

Decentralized Fair Bandwidth Assignment

Priority-based communication buses can be used to connect one or more bandwidth streams (cf. Section 2.1.3). However, the problem that arises when using static priorities to arbitrate the access can be characterized by the example shown in Figure 3.1. Two independent nodes try to transmit video streams to a central displaying node. Each stream tries to send as many messages as possible in order to increase the video quality. Due to the layout of the communication protocol, one of the streams has a higher priority than the other. The stream with higher priority increases the amount of transmitted messages. The stream with lower priority is not able to transmit because no bandwidth is remaining.

This chapter is organized as follows. In Section 3.1 the problem is defined and placed into the context of related work. In Section 3.2, the priority-based medium access game is formulated, which represents the fundamentals of the proposed methods. Section 3.3 presents the distributed reinforcement learning algorithm to establish fair bandwidth distribution. The parameters are analyzed and it is proven that the algorithm also works together with other scheduling methods. In Section 3.4, the *period access scheme* is introduced and analyzed. Finally, in Section 3.5 the developed learning algorithm is tested in a CAN simulation.

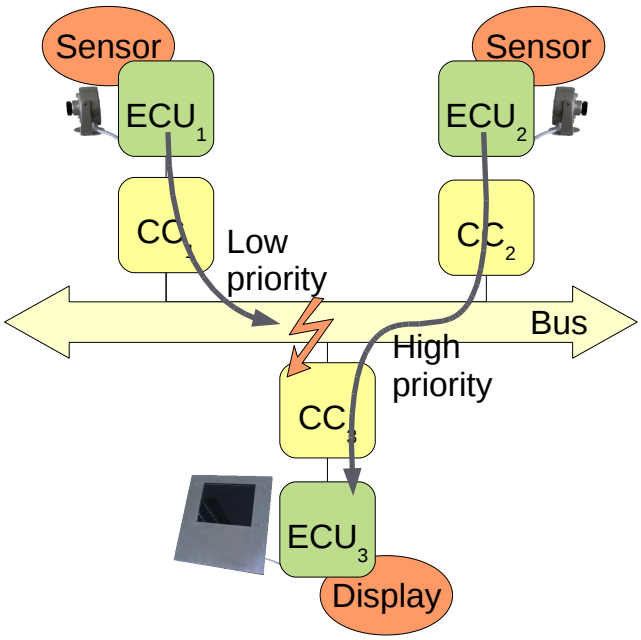


Figure 3.1: Example scenario where two video streams are connected to one bus.

3.1 Problem Definition and Related Work

The problem tackled in this chapter is how to use communication resources as efficient as possible. The main problem is the contradicting goals of, on the one hand, guaranteeing timing requirements and, on the other hand, using the full data rate of the available medium, i.e., not wasting resources. As already described, we are targeting communication resources with fixed priority-based access scheme. Even though our approach is applicable for any priority-based access mechanism, we will use the CAN protocol as an example throughout this thesis.

In Section 2.1, we distinguish between different requirements of the streams. In this chapter, an approach concentrated on bandwidth streams is presented, where it is not important to meet deadlines but to send a certain amount of messages per time unit. A related model for such applications is the elastic model presented in [BLCA02]. It assumes a periodic task occurrence with a specified minimal and maximal period. To control the adaptation of the periods the elastic coefficient is introduced, which represents the flexibility of a task to change its period. We generalize this definition by also considering applications with a probabilistic task occurrence, but at the same time assume that the elastic coefficient is equal for everyone. Additionally, we focus on applications, where it is possible that the quality of the application varies with the bandwidth

it can use, so the requirement is to transmit with a data rate as high as possible. Concretely, we solve the problem of sharing the available bandwidth fairly.

Examples of our target applications can be found in the automotive industry and industrial automation. For example video streams fulfill perfectly our assumptions. In Chapter 5, we introduce a method to enable the streaming of video over CAN. Video streams are throughput sensitive, which means the amount of data transmitted is more important than the delay. Furthermore, depending on the available bandwidth the quality of the video stream can be adjusted in several ways. The easiest would be to transmit as many frames as possible, so if the available bandwidth is reduced the frames per second are reduced. Another way would be to increase the compression rate or lower the resolution. In the automotive domain, cameras that are mounted at positions where the driver has no line of sight can increase the safety. Furthermore, gathering information for intelligent driver assistance systems, such as described in [GT06] will increase the need for streaming videos through the vehicle. In industrial automation, using the already available CAN for video surveillance can safe wiring cost.

The importance of self-organizing bandwidth sharing will increase in future real-time systems as the complexity grows. For example, [DZDN+07] presents an automotive architecture where 29 ECUs are connected in one priority-based network. One of the functions of the distributed ECUs is to transfer information from 360° sensors to several actuators. These sensors have equal importance to the quality of the control mechanisms. This means that additional information from one of the sensors is useless, if the other sensors cannot provide their data as well. Especially, when increasing the sampling rate of the sensors or adding additional components, the complete system has to be re-designed. However, the priority-based access does not allow to assign equal priority to all sensors.

In this chapter, we use game theoretic principles to develop our algorithms. Game theory is a branch of mathematics that can be used to analyze learning in multi-agent systems [BBDS08]. It has been often applied to study behavior of computer networks since it offers a framework to model and analyze situations in which players have to make decisions that have mutual, possibly conflicting consequences. In recent years, several game theoretical approaches to model and understand the dynamics of medium access and communication over an exclusively shared medium have been made [CCLD07, FB04, HRGD05]. The main focus of this research lies on contention-based medium access of Carrier Sense Multiple Access with Collision Detection (CSMA/CD). Such approaches use collision detection mechanisms: whenever a collision is detected, all transmissions are aborted and the senders wait for some random time before trying to transmit data again. Especially, selfish (i.e., greedy) behavior is analyzed. The problem is that the more often nodes try to send data, the more collisions may occur and less amount of data can be transmitted in the end. The purpose of

the game theoretic analysis is to find protocols for medium access to overcome this problem. Further goals are to distribute bandwidth fairly [RG05] or to reduce latency and meet deadlines [FMN07]. The equilibria of these protocols are analyzed which are commonly reached when all nodes try to maximize their profit simultaneously (known as Nash Equilibria [Nas51]). Results show that when designing protocols properly, the aforementioned goals can be achieved.

Besides game theoretic approaches, Cena and Valenzano worked on alleviating the pure priority-based access of CAN. In [CV95], a distributed mechanism to improve fairness in CAN is proposed. They use especially reserved identifiers to update a distributed queue. The queue defines the order in which the nodes get access to the bus. However, they need to use the *extended identifier* [CAN91] of the CAN protocol. This extension increases the size of the identifier from 11 Bits to 29 Bits and the overhead for each message from 42 Bits to 62 Bits. A distributed round-robin access is established with the protocol presented in [CV02]. In contrast to our work presented in this chapter, the goal was to provide similar response times to certain groups of messages while we try to provide similar bandwidth. Furthermore, other negotiation mechanisms, such as token passing or client-server architectures, could be used. These would also need additional communication, which is a scares resource in distributed embedded systems. In contrast, the computation capabilities of the nodes increase with the advances in miniaturization of transistors. Therefore, our approach uses distributed, passive monitoring of the current traffic to achieve a global fair bandwidth distribution without any communication overhead.

3.2 Game Theoretic Analysis of Medium Access

In this section, we describe how game theory can be used to model priority-based medium access. This is an adaptation of the Contention-based Medium Access Game described in [RG05].

The fundamental idea of game theory is to describe a system or a problem by a *game*. In general, a game G consists of a set of k players $K = \{1, 2, .., k\}$, a set of strategies A_i available for each player i, and a specification of payoffs for each combination of player strategies. For an individual player i, the payoff is given by the utility function $F_i(\mathbf{a})$ where $\mathbf{a} = (a_1, ..., a_k)$ represents the strategies chosen by all players.

If all players have reached a strategy $a_i \in A_i$ in which no one benefits from changing its strategy, an *equilibrium* is reached. A strategy vector $\mathbf{a} = (a_1, ..., a_k)$ is called *Nash equilibrium* [Nas51] if no player can get a better response by changing its strategy unilaterally. This means that all players try to maximize their utility function while all other players keep their strategy. Defining a strategy vector $\mathbf{a}_{-i} = (a_1, a_2, ..., a_{i-1}, a_{i+1}, ..., a_k)$ that contains the

strategies of all players aside from player i, the Nash equilibrium can be formalized as follows.

Definition 3.1. A strategy vector $\mathbf{a}^*$ is a *Nash equilibrium* iff

$$F_i(a_i, \mathbf{a}^*_{-i}) \leq F_i(a^*_i, \mathbf{a}^*_{-i}), \forall a_i \in A_i, \forall i.$$

For further analysis, we assume that a system is consisting of several streams that want to access a shared medium. Each stream has exactly one predefined priority. On a physical computation unit, for example a micro controller, several streams can be scheduled. Even though streams on the same computation unit could easily exchange information, due to flexibility and simplicity reasons we consider them independent. Because we assume the arbitration within the physical computation unit is also priority-based, the behavior will be the same. In the following, we use the terms *stream* and *player* interchangeably, while stream points out the physical aspect, i.e., one CAN priority, and player the game theoretic one.

Such a medium access can be described using the *Normal Form*, which is defined as follows: for each strategy combination, the expected payoff of each player is given. For a two player game, this Normal Form can be defined as a matrix as shown in Figure 3.3. Each cell contains the payoff $(F_1(a_1, a_2), F_2(a_1, a_2))$ of the players strategy dependent on the strategy selected by the other player. The strategy of player 1 is listed in the row, the one of player 2 in the column.

3.2.1 Medium Access Game

The medium access can be divided into rounds. Each round corresponds to one bus access and is divided into two phases: the arbitration phase and the transmission phase. This is illustrated in Figure 3.2. In priority-based communication systems, each participant has a unique priority. During the arbitration phase, the node with the highest priority that wants to send data is chosen. In the transmission phase, this node gets access to the shared medium. Using this concept, collisions can be resolved. An example is shown in Figure 3.3. Here, we assume that player 1 has a higher priority than player 2. If only one player wants to access the medium, it gets a grant and the payoff is 1. A collision occurs when both players want to send. In this case, the player with the highest priority (here player 1) gets the grant. As defined in Section 2.1, players with a lower index have a higher priority.

For the game theoretic analysis, the following assumptions about the system are made:

1. Each player has always one message to send.
2. Each player has a message of equal length.

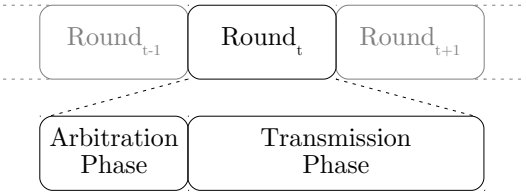


Figure 3.2: The two phases, *arbitration phase* and *transmission phase*, that define the rounds of the priority-based Access Game.

		player 2	
		wait	send
player 1	wait	0, 0	0, 1
	send	1, 0	1, 0

Figure 3.3: Two player priority-based Medium Access Game in Normal Form

3. The system is stable, so no technical failure occurs.

In later sections the consequences for loosening the following assumptions will be evaluated: assumption 1 in Section 3.3.3 and Section 3.4, and assumption 2 in Section 3.5.

In order to achieve fair bandwidth sharing, a *mixed strategy* for each player has been introduced in [RG05]: Instead of having the two discrete strategies *send* and *wait*, a probability distribution on strategy space A_i is given. Because our strategy space is limited to two strategies, the strategy can be reduced to a single value $p_i \in [0, 1]$. This means that player i sends with probability p_i and waits with probability $1 - p_i$.

Definition 3.2. The *priority-based Medium Access Game* G_{mixed} is defined as a tuple $G_{mixed} := (K, F_i)$ with $i \in K$ where

- $K = \{1, ..., k\}$ is a set of k players.
- $F_i(\mathbf{p})$ is the utility function for player i dependent on the strategies chosen by all players $\mathbf{p} = (p_1, ..., p_k)$.

The utility function for a player i is equal to the probability to successfully transmit data. Player i successfully sends a message if he chooses to send and

all players with higher priority decide to wait. The nodes with lower priorities do not have influence on the payoff. Thus, the utility function is given by:

$$F_i(\mathbf{p}) = p_i \cdot \prod_{j<i}(1 - p_j). \tag{3.1}$$

Now, we want to give an overview how the priority-based Medium Access Game can be used to define and analyze fair bandwidth sharing. Detailed proofs can be found in [WZT09*]. Potentially, there are many different ways to define fairness. For example in computer networks the maxmin fairness is a generally accepted concept [BGH92]. However, a quantification of the degree of fairness is impossible. Therefore, another widely used fairness measure is the fairness index introduced by Jain et al. [JCH84]. The commonality for all measures is that the maximum value is reached when all parties receive the same amount of resources. For this reason, we define fairness for the application presented in this paper as follows.

Definition 3.3. A strategy vector $\widetilde{\mathbf{p}}$ is called fair if each player has the same probability to get access to the shared medium. This means that the following constraint always holds:

$$F_1(\widetilde{\mathbf{p}}) = F_2(\widetilde{\mathbf{p}}) = ... = F_n(\widetilde{\mathbf{p}}) \tag{3.2}$$

We now want to consider a medium where $b \in [0, 1]$ determines the available bandwidth. Then a strategy $\widetilde{\mathbf{p}}$ is fair if each player gets $\alpha = \frac{b}{k}$ bandwidth ($F_i(\widetilde{\mathbf{p}}) = \alpha$). To achieve this fairness, each node i has to choose as sending probability:

$$\widetilde{p_i} = \frac{b}{k - (i-1) \cdot b} \tag{3.3}$$

The idea of choosing these probabilities is best explained by an example. A detailed proof why this distribution leads to fair bandwidth sharing can be found in [WZT09*]. For simplification let $b = 1$. Then, Equation (3.3) reduces to $\widetilde{p_i} = \frac{1}{k-i+1}$. Let's assume a scenario with two players ($k = 2$) with the probability distribution $p_1 = 0.5$ and $p_2 = 1$. Consequently, player 2 always tries to access the medium and player 1 tries every second round. Because of the priorities, the players are granted access alternately. In general, the idea is that the higher the priority the more rare the node tries to send data. Players with lower priorities try to send more often because they are blocked by higher priorities if they want to transmit data concurrently.

3.2.2 Enhanced Priority-based Medium Access Game

Equation (3.3) describes the sending probability each player has to chose to reach fair bandwidth sharing. To calculate this strategy, each player needs global

information, i.e., how many players join the game k and where its priority is ranked among them, given by i. This is very inflexible in case new players are added or removed. We want to develop a *self-organizing multi-agent system* that can solve the problem of fair bandwidth sharing with just local information.

Local information available for a node can be obtained by observing the bus traffic. In the proposed game theoretical model, the traffic information corresponds to the utility a player receives. The example in Figure 3.3 shows that the dominant strategy for player 1 is to always send. The problem is that streams with lower priorities cannot influence the strategies of higher-prior ones. Furthermore, there is no information available for a higher-prior stream that a lower-prior one tries to access the medium and gets blocked. Therefore, the medium access is extended by introducing a new constraint that claims that a minimal amount of bandwidth ϵ has to stay free. In case this constraint is not satisfied, the players are penalized, enforcing them to change their behaviors. This allows players with small priorities to influence the outcome of the game.

This *enhanced priority-based medium access game* is described formally the following way. The probability that the medium is free p_{free} is equal to the probability that all players decide to wait:

$$p_{free} = \prod_{j \in K} (1 - p_j) \tag{3.4}$$

The constraint that a minimal amount ϵ of the bandwidth stays free implies that the condition $p_{free} \geq \epsilon$ is fulfilled. If this condition is not fulfilled, all players are punished by setting their utilities to 0. This has a direct influence on the utility function as follows:

$$F_i(\mathbf{p}) = \begin{cases} p_i \cdot \prod_{j<i} (1 - p_j), & \text{if } \prod_{j \in K} (1 - p_j) \geq \epsilon \\ 0, & \text{else} \end{cases}$$

When all players try to maximize their utility function, the system ends in a Nash equilibrium. In such a system state, no player has an intent to switch the strategy because the payoff will not increase if the others do not change their strategy. In [WZT09*], it is proven that all players $i \in K$ are in a Nash equilibrium *iff* choosing strategies $\bar{\boldsymbol{p}}$ that fulfill

$$\prod_{i \in K} (1 - \bar{p}_i) = \epsilon.$$

It can furthermore be shown that the fair strategy of Equation (3.3) fulfills this property and is consequently a Nash equilibrium. This means that as soon as the players reach this fair strategy no player has an appeal to differ from its strategy. These results imply that it is possible to provide learning methods

for the enhanced game to establish emergent and self-organizing organic methods for bus-based communication architectures. In the following, we provide adequate learning algorithms.

3.3 Probability Access Method

The theoretic analysis from Section 3.2.2 suggests to learn a fair strategy. A good overview on learning in games can be found in [FL98]. One way is reinforcement learning, a learning method where the chosen actions are based on rewards that were received for previous actions. It is very well suited for learning games since the reward for a certain action is defined by the games. As multiple players are involved, the research field is called *multi-agent reinforcement learning*. A comprehensive survey can be found in [BBDS08, BBD06].

According to [BBDS08],our priority-based access game falls into the field of *fully cooperative static games*. Fully cooperative means that the players try to jointly reach a common goal - In our case: fair bandwidth sharing. It is a static game, because for a given set of actions, the reward for each player is always the same and is not depending on any foreign environment or a system state. For these type of games, a centralized controller could learn an optimal joint-action using Q-learning [WD92]. However, when the players are independent decision makers, a coordination mechanism has to be introduced. Several communication-free methods have been developed to avoid this communication overhead [Lit01, LR00]. The problem with these algorithms is, that they assume that each player knows the rewards of all other players. In our scenario, this would cause additional amount of communication and violate our idea of information hiding among players. For this reasons, a new game-specific algorithm is presented in this section.

3.3.1 Penalty Learning Algorithm (PLA)

As a first step, we present an algorithm that learns the fair strategy using the assumptions from the enhanced priority-based access game. This means, the used amount of bandwidth is controlled by accessing the media at each round with a probability p_i, i.e., strategy. The Penalty Learning Algorithm (PLA) works as follows: Each stream i adapts its strategy p_i according to Algorithm 3.1 after monitoring a certain time interval as shown in Figure 3.4. The goal of the distributed algorithm is to reach a global fair bandwidth distribution with only local knowledge of each stream.

In detail this means: When playing the priority-based medium access game as described in Figure 3.2, after l_{mi} rounds the sending probability p_i of each player i is locally updated. This update is called a *learning step* and a pseudocode

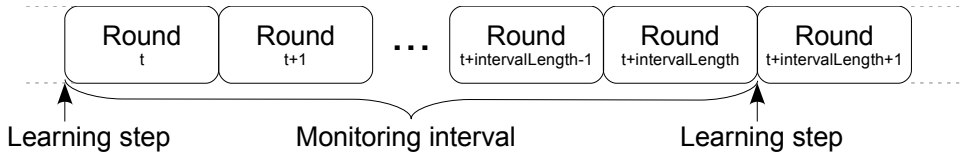


Figure 3.4: PLA illustration - every l_{mi} rounds a learning step is performed

Algorithm 3.1 Learning step of PLA for each player

1: $load = \frac{m_{sent}}{l_{mi}}$;
2: $success = \frac{nSuccess}{l_{mi}}$;
3: **if** $(load > 1 - \epsilon)$ **then**
4: $\quad \Delta = -success$;
5: **else**
6: $\quad \Delta = (1 - success)$;
7: **end if**
8: $p_i^{(t+1)} = p_i^{(t)} + \eta \cdot \Delta$;

description is given in Algorithm 3.1. First the *load* of the last *monitoring interval* has to be calculated (line 1) by dividing the number of rounds in which a player has sent a message m_{sent} by the total number of rounds of the monitoring interval l_{mi}. Additionally, the number of successful transmitted messages *success* (line 2) denotes the number of rounds player i has successfully transmitted $nSuccess$ in the monitoring interval divided by the total number of rounds l_{mi}. The learning algorithm works by adapting the sending probability according to the amount Δ calculated in line 3 to 6. Depending on the load of the monitoring interval, the sending probability is decreased (*load* is greater $1 - \epsilon$), or increased (*load* is less or equal to $1 - \epsilon$). The amount depends on the success of the player. The more success the player had, the more it will decrease its sending probability in case of penalty and the less it will increase the sending probability in case of no penalty. This basic mechanism makes it possible that the players emerge to a fair bandwidth distribution. The change of the sending probability Δ is multiplied by a learning rate $\eta < 1$.

To explain the functional principle of this distributed learning algorithm, we demonstrate the behavior of the algorithm on a two player example. Using the graphical Normal Form representation of the game, we can explain why the algorithm will converge to a fair bandwidth distribution. The enhanced priority-based access game cannot be illustrated by a two action Normal Form as in Figure 3.3, because the reward calculated in a single round is either one or zero. Instead, a more fine grained bandwidth is needed to use a reasonable ϵ. In consequence, it only makes sense to play a mixed strategy. If the game is played for several rounds, the free bandwidth can be calculated by dividing the

number of free slots by the number of rounds played. If this number of played rounds reaches infinity, then the free bandwidth can be calculated as described in Section 3.2.1.

In Figure 3.5, the graphical Normal Form of the enhanced two player priority-based access game with $\epsilon = 0.3$ is shown. On the left column, the strategy of player 2 is displayed and in the top row, the actions of player 1. The strategy in this case is the probability p_i that player i is trying to send during each round. The probability p_i of course is a real number between 0 and 1, but for demonstration purpose we display only 11 possible strategies per player. The remaining entries show the *success*, i.e., the amount of bandwidth the players get when playing the strategy, the first number for player 2 and the second for player 1. The green (semi-dark) and yellow (light) entries in the upper left part denote the strategies where the amount of free bandwidth is below ϵ and therefore the reward F_i is equal to *success*. The red (dark) entries denote the strategies where the ϵ -rule is violated and the reward is zero. The values within the table cells represent the value of *success*, because it is used by the learning algorithm. The yellow (light) entries show Nash equilibria, where a standard game learning algorithm would converge to. It shows that many unfair Nash equilibria do exist, which we try to avoid.

In the example, we assume $\eta = 0.2$. A detailed discussion of the learning rate η is presented in Section 3.3.2.1. The thin small arrows in the figure show the change of action, i.e., the gradient of the sending probability, that is applied according to the PLA given in Algorithm 3.1. The four large arrows show the tendency of all gradients. The tendency towards the Nash equilibria front, represented by the top left and bottom right arrow, can be easily seen. The tendency towards the fair solution on the front needs further explanation. The tendency from bottom left to top right can be read when looking at the arrows on the lower half of the Nash equilibria front. For example the resulting direction from arrow 1 and 2, depicted by the white arrow, tend towards the fair solution and so do the other combined arrows in this region. When looking at the top half of the Nash equilibria front, the same conclusions can be drawn as for the lower half. As an example the resulting direction from arrow 3 and 4 are depicted by the white arrow and tend towards the fair solution.

Once reaching the fair solution, the strategies stay there. This is clarified by looking at arrow 5 and 6 close to the fair solution. They are parallel and point in opposite direction. So once reached this point, the strategies are oscillating around this point with a maximal length of η.

3.3.2 Experimental Evaluation

In the following, we simulate the behavior of the PLA. Our simulation has five degrees of freedom: starting probability of each player p_i^{start}, free bandwidth ϵ,

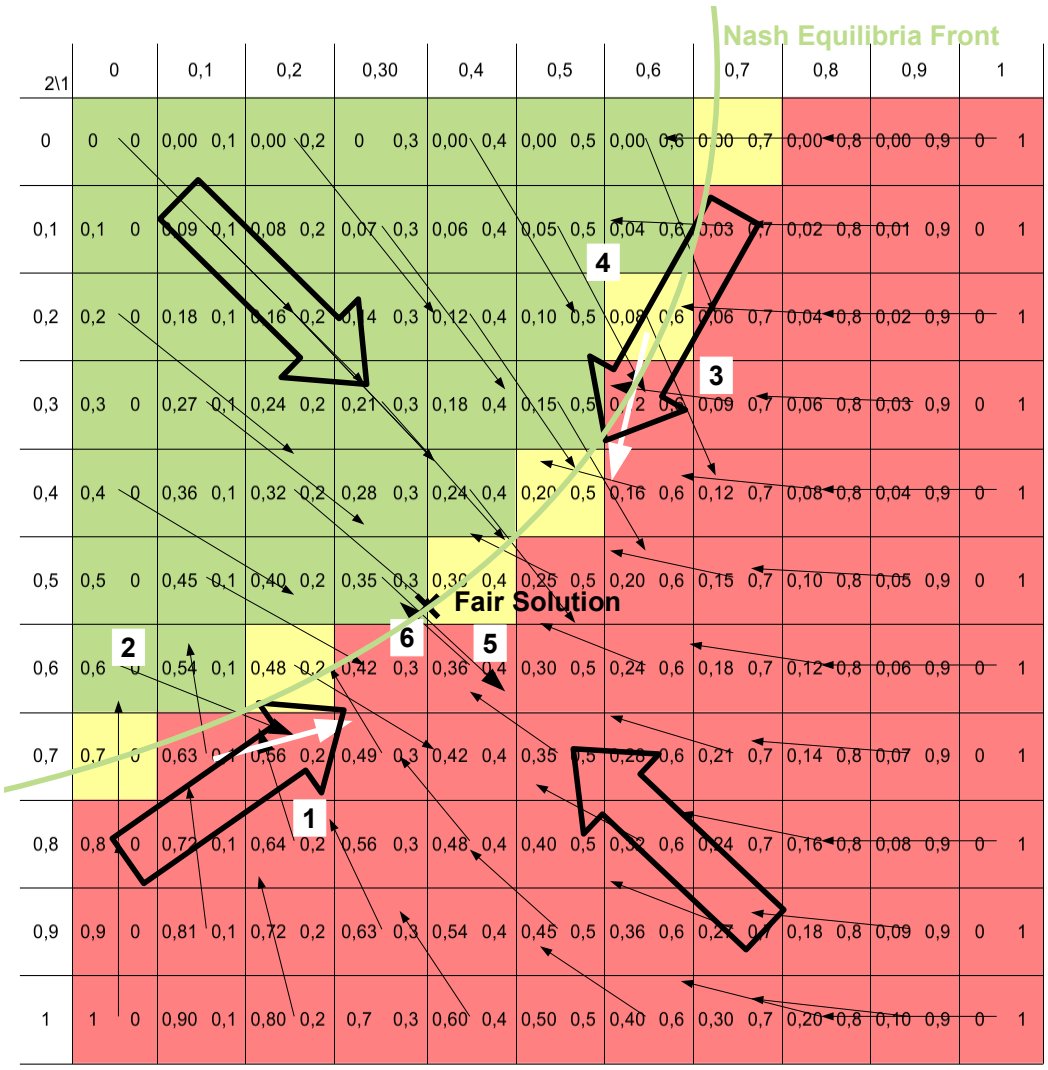


Figure 3.5: Graphical Normal Form representation of the two player enhanced priority-based access game with $\epsilon = 0.3$

number of players k, l_{mi}, and learning rate η. The starting probability of each player is always set to $1-\epsilon$, because experiments showed that it doesn't influence the converged state and with a fixed initial value, we can better compare different settings. Additionally, this represents the case where the players choose the most greedy strategy, where they would receive the highest reward when playing alone.

The value of ϵ is depending on the implementation. It should be set to the smallest possible value to waste as few bandwidth as possible. In our simulation,

$\epsilon = 10^{-15}$ is used. For all our experiments this is several orders less than one round. This means, when playing fair the whole bandwidth is used.

As this is a multi-agent game, the most simple example is to have two players. In the following, we assume that the lower the player id the higher is the priority. The l_{mi} is first chosen quite large to ensure the law of large numbers is valid and the reward is very close to our theoretical calculations. The learning rate η will be analyzed in the next section.

During each simulation time step, the bandwidth is displayed to show the convergence and the stability of our algorithm. The performance of the algorithm is described by the following two quality criteria:

Definition 3.4. The *convergence time* $t_{converge}$ is the time needed to reach the converged state. The converged state is reached when the accuracy doesn't change anymore.

Definition 3.5. The *accuracy* μ is the maximum amount that the used bandwidth deviates from the fair bandwidth in the converged state. The *fair bandwidth* α is the bandwidth used by all players when playing a fair strategy.

The faster a fair solution is reached, i.e., the smaller the convergence time, the better. In the following experimental analysis, we make following assumptions:

- The used bandwidth is similar to the variable *success* in Algorithm 3.4.
- The accuracy is defined by:

$$\mu = max(|\alpha - success_i(t)| \quad \forall t \in]t_{converge}, t_{sim}], \forall i \in K)$$

- t_{sim} is chosen by experiments until the behavior in the converged state repeats.
- For a two player game the fair bandwidth is $\frac{1}{2}$, because ϵ can be neglected.
- A simulation time step equals one played round and therefore, the terms will be used interchangeably.
- The plotting interval defines the interval used to calculate the current bandwidth used.
- If not written otherwise the simulation parameters from Table 3.1 are used.

Parameter	Description	Value
p_i^{start}	Starting probability	$1 - \epsilon$
ϵ	Free bandwidth	10^{-15}
k	Number of players	2
l_{mi}	Length of monitoring interval	$100,000$
η	learning rate	0.2

Table 3.1: Default simulation parameters

3.3.2.1 Analysis of learning rate η

Figure 3.6 shows the simulation results for $l_{mi} = 100,000$ and $\eta = 0.2$. It can be seen that the algorithm converges in 5 learning steps, i.e., 500,000 time steps, but then oscillates with an accuracy of $\mu = 0.05$. If $\eta = 0.02$, then the convergence time is 150 learning steps as shown in Figure 3.7. In Figure 3.8 the mean and the variance of the bandwidth are shown. Here, we can see that the full bandwidth is used from the beginning, because the mean is always 0.5. Additionally, it shows that the variance decreases as the system reaches the converged state. Figure 3.9 shows the access probabilities p_i of the two players. Player 1 keeps its strategy, while Player 2 is adjusting to the fair probability of 0.5. At this converged state, the accuracy is 0.018. The convergence time is inversely proportional to η, and the accuracy is directly proportional to η, as shown in Figure 3.10. In conclusion, the first experiments show that the algorithm converges and that there is a tradeoff between the convergence speed and the accuracy.

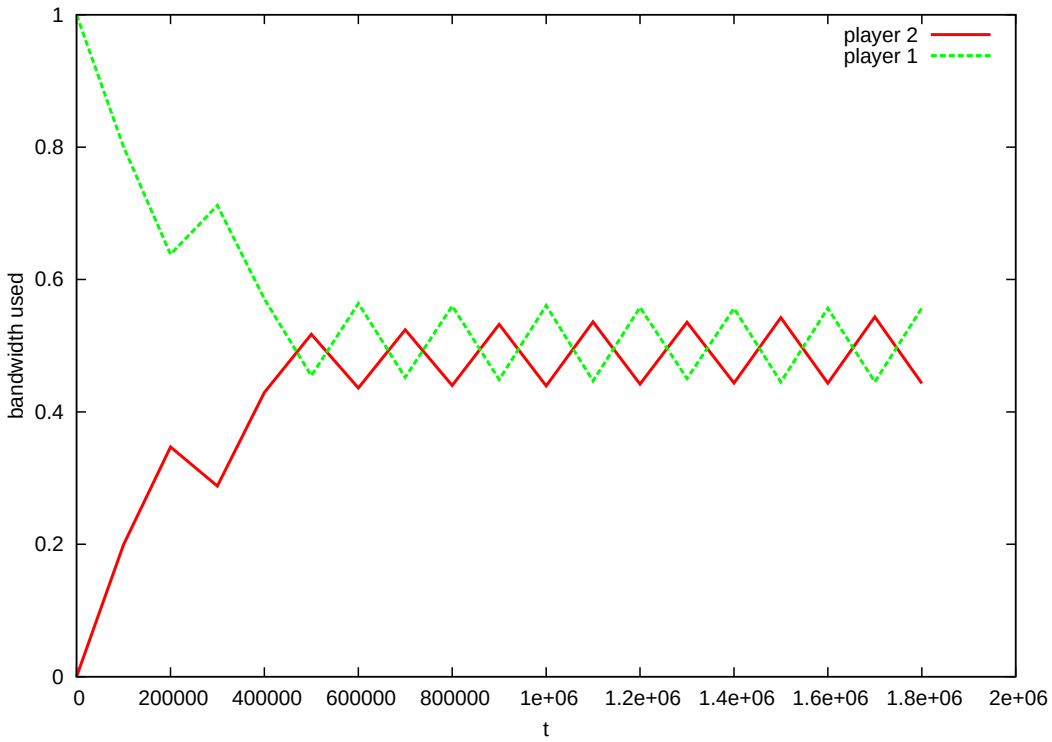


Figure 3.6: Simulation results of the penalty learning algorithm for $\eta = 0.2$

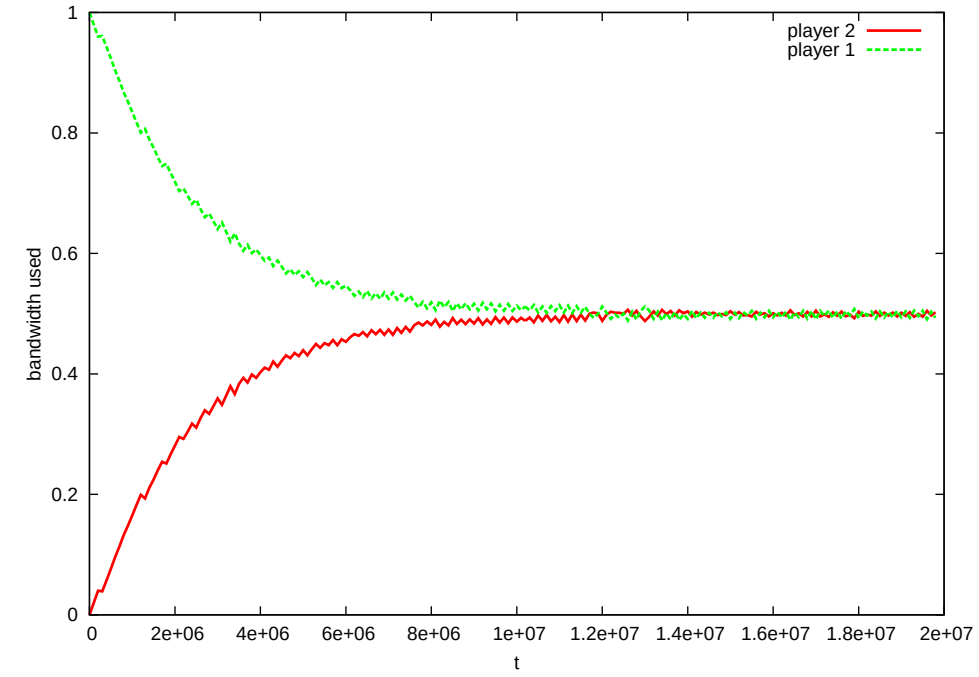


Figure 3.7: Used bandwidth of both players for simulation of the penalty learning algorithm for $\eta = 0.02$

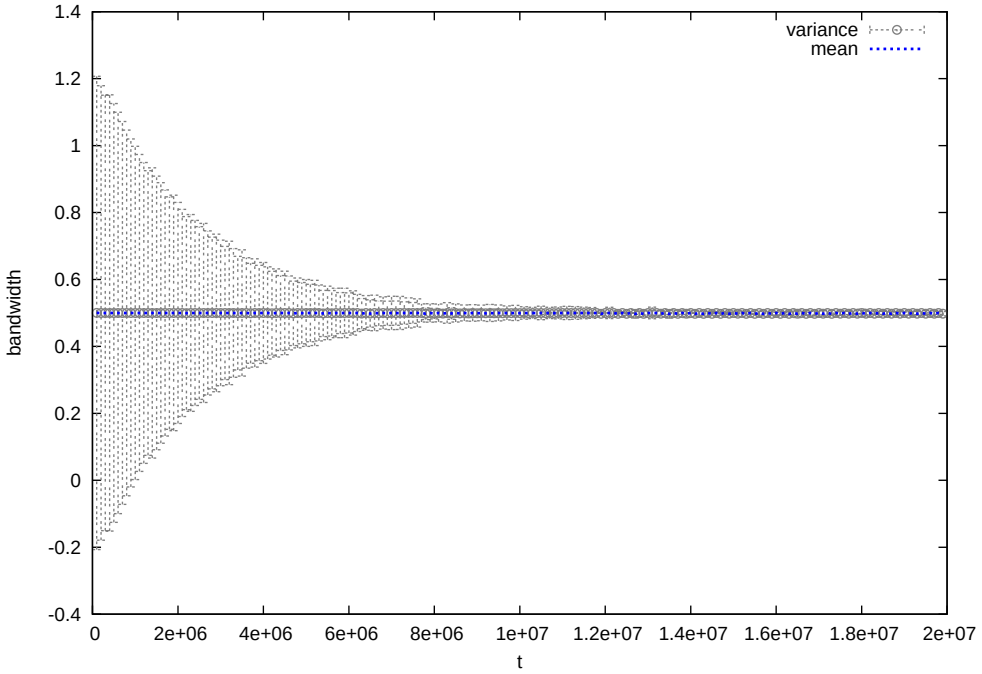


Figure 3.8: Mean and variance of used bandwidth for simulation of the penalty learning algorithm for $\eta = 0.02$

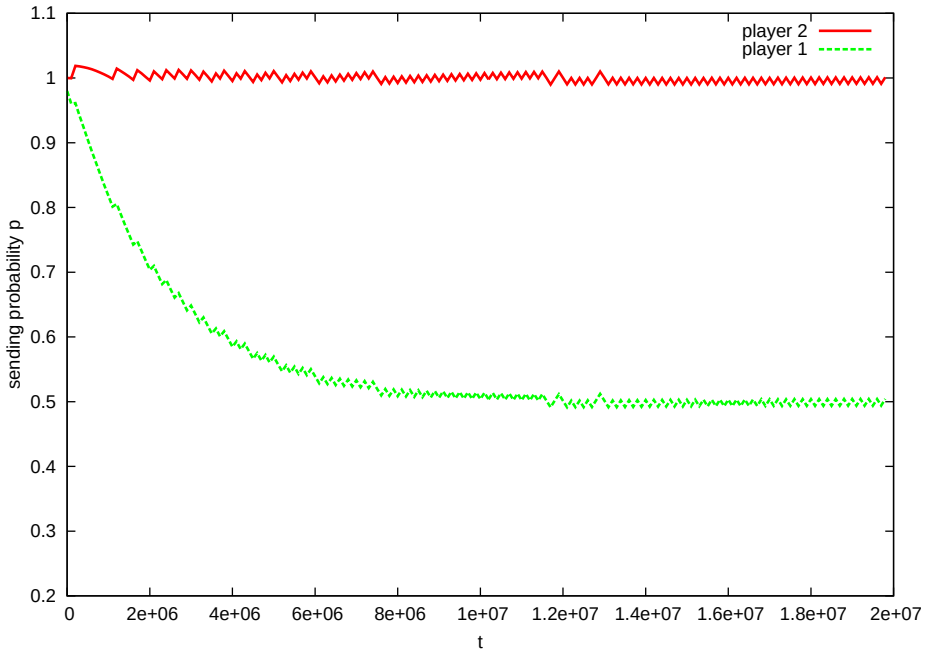


Figure 3.9: p of both players for simulation of the penalty learning algorithm for $\eta = 0.02$

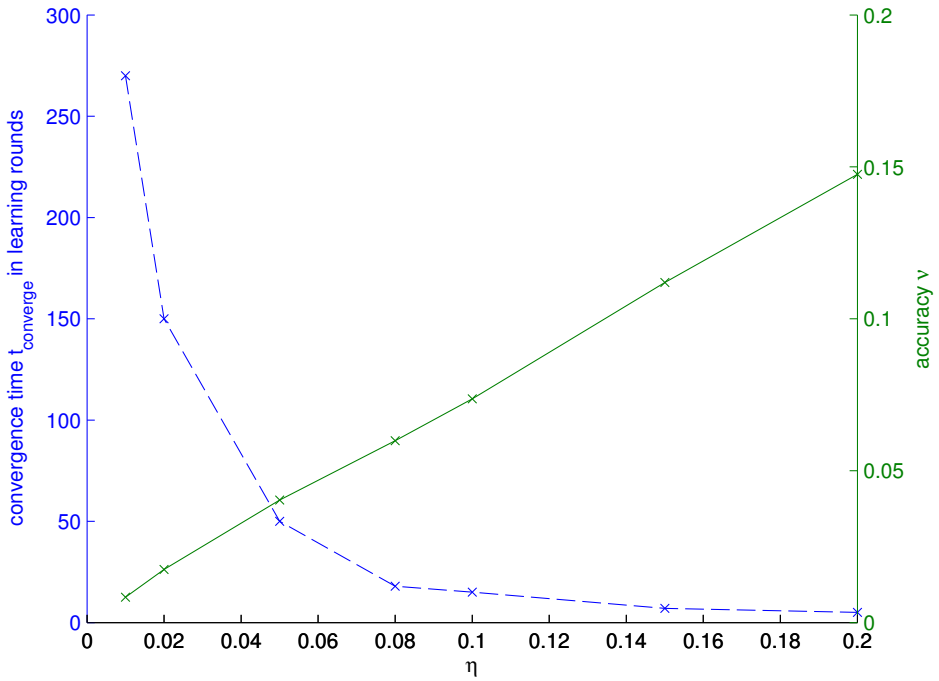


Figure 3.10: Convergence time and accuracy of the penalty learning algorithm for different η

3.3.2.2 Analysis of the number of players k

In the following, we want analyze the behavior when the number of players is increased. Figure 3.11 shows the simulation results of the priority-based medium access game for $k = 10$ players. Compared to the two player scenario in Figure 3.7, the convergence time increases. This is in the nature of the priority-based access mechanism, because the high-priority messages get full access and need to learn by penalty that other low priority messages exist. However, after reaching the converged state, the accuracy is similar to the two player scenario ($\mu = 0.015$). Looking even further, with 20 players, as shown in Figure 3.12, the same behavior can be observed: The time to converge increases, but the accuracy in the converged state stays equal ($\mu = 0.014$). This shows the scalability of our approach.

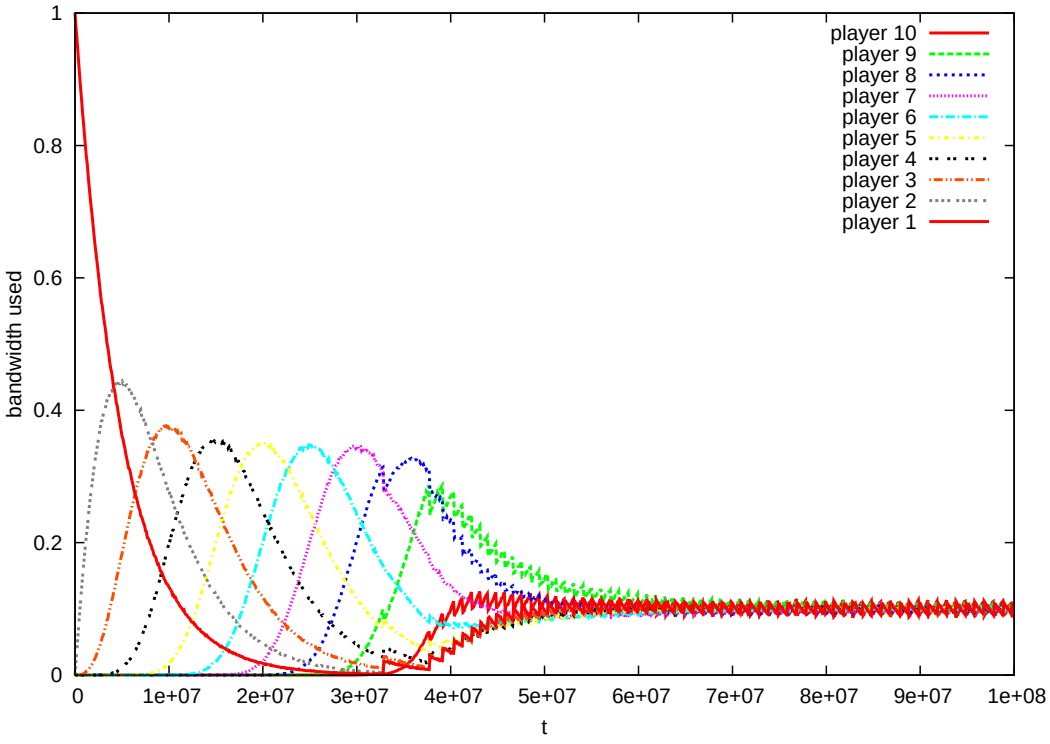


Figure 3.11: Simulation results of the penalty learning algorithm for 10 players and $\eta = 0.02$

3.3.2.3 Analysis of the length of the monitoring interval l_{mi}

In order to analyze the length of the monitoring interval l_{mi} in terms of convergence time and accuracy, we will again compare the following results to the

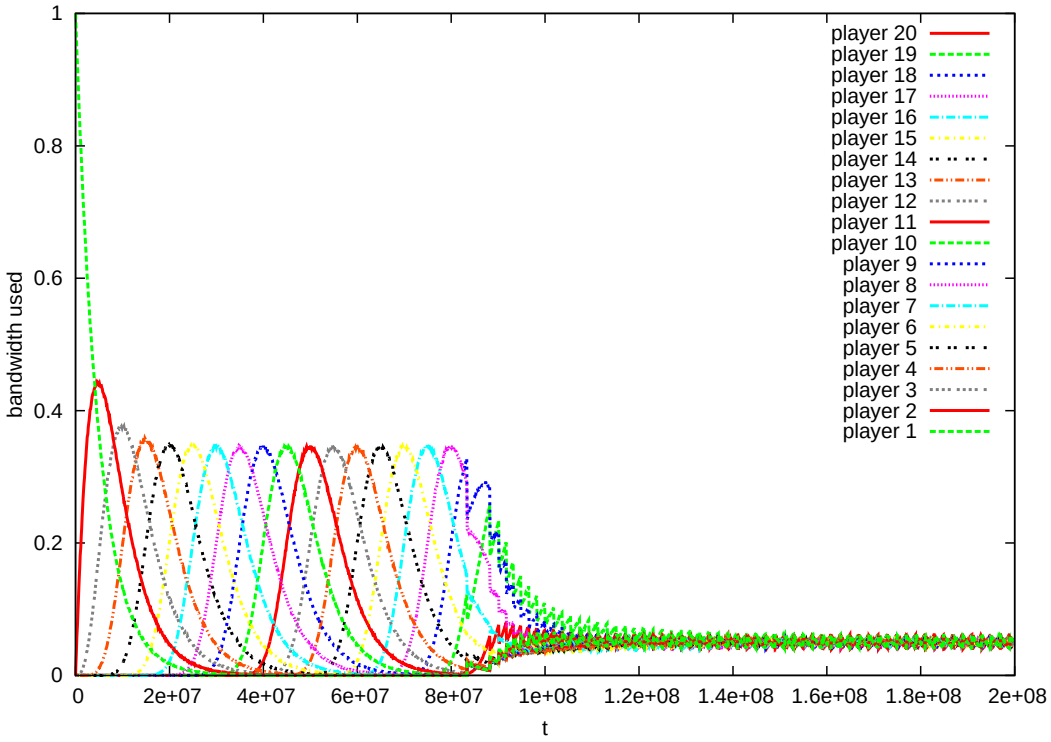


Figure 3.12: Simulation results of the penalty learning algorithm for 20 players and $\eta = 0.02$

two player scenario in Figure 3.7 and therefore use $\eta = 0.02$. On the previous plots, we always plotted the bandwidth of the players by dividing the number of successful messages during one l_{mi} by l_{mi}, which means the plotting interval is equal to the monitoring interval. If we decrease now l_{mi}, the influence of the actions chosen can differ a lot from the resulting bandwidth distribution. This effect can be seen, when two players are behaving fair from start with $l_{mi} = 100$. Two fair players on the priority-based access game have $p_1 = \frac{1}{2}$ and $p_2 = 1$. The simulation results are shown in Figure 3.13. However, for many applications, this short-term variations can be neglected and only the average over several messages is interesting. For example, for a video stream it is only interesting when the next full frame has been transmitted which consists of many messages.

Apart from the difference between action and result, two aspects for lowering l_{mi} have to be considered:

1. the amount of computation and

2. the *load*-granularity.

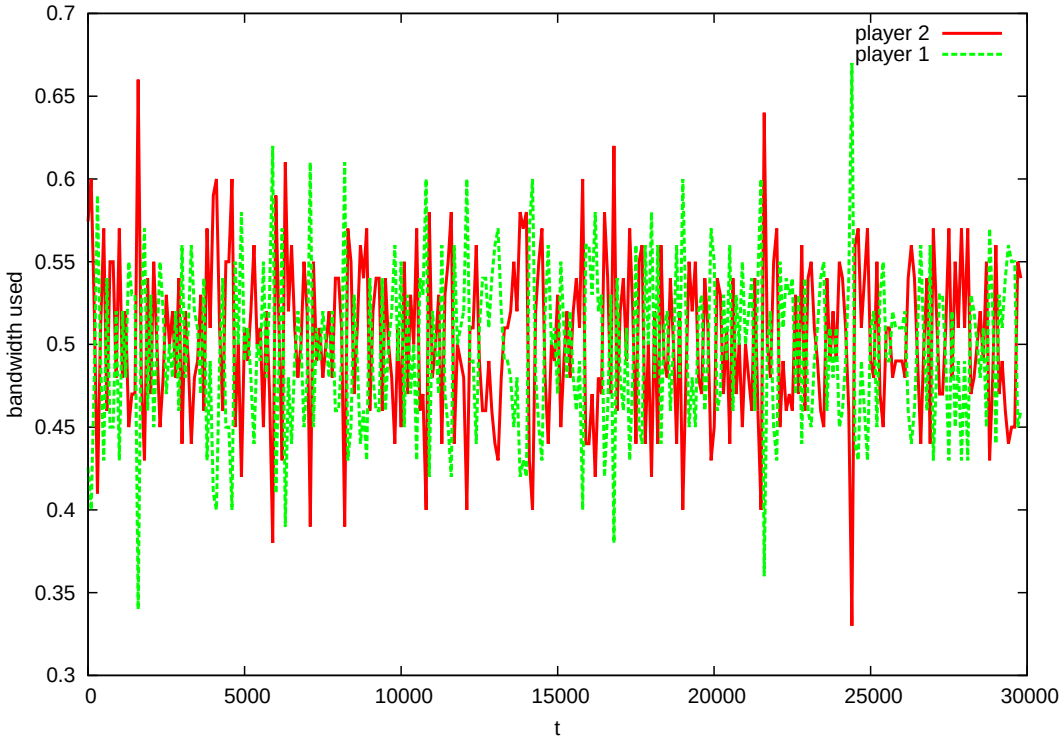


Figure 3.13: Simulation results for two fair players from start with $ilength = 100$

Lowering l_{mi} means that learning is done more often. This in consequence requires more computation time. So, when looking at the following results we always have to keep this in mind.

The *load*-granularity effect is as following. The PLA decides whether to lower or to raise the sending probability on the *load* during the last monitoring interval. Therefore, it is essential that the *load* can take enough values between 0 and 1. For example, if we set $l_{mi} = 1$, the *load* would only take the values 0 or 1. So, no matter what ϵ is set to, the algorithm will raise the probability if no one sends and lower it if anyone sends. The results with $l_{mi} = 1$ and a plotting interval of 100 are shown in Figure 3.14. Amazingly, the system converges very fast to a state where the bandwidth of each player is $\frac{1}{n+1}$. This behavior can be observed for any number of players. However, as our intended bandwidth for each player should be $\frac{1}{n} > \frac{1}{n+1}$, this behavior is not further investigated.

Figure 3.15 shows the simulation results for the penalty learning algorithm for $l_{mi} = 100$. Compared to the simulation from Figure 3.7, even though the variance is much higher, the convergence time is about 60 learning steps, i.e., 6,000 time steps lower. As l_{mi} is much shorter, the total number of time steps to converge is reduced tremendously from 15,000,000 to 6,000. The accuracy in the converged state is also improved to 0.001 compared to 0.018, because the

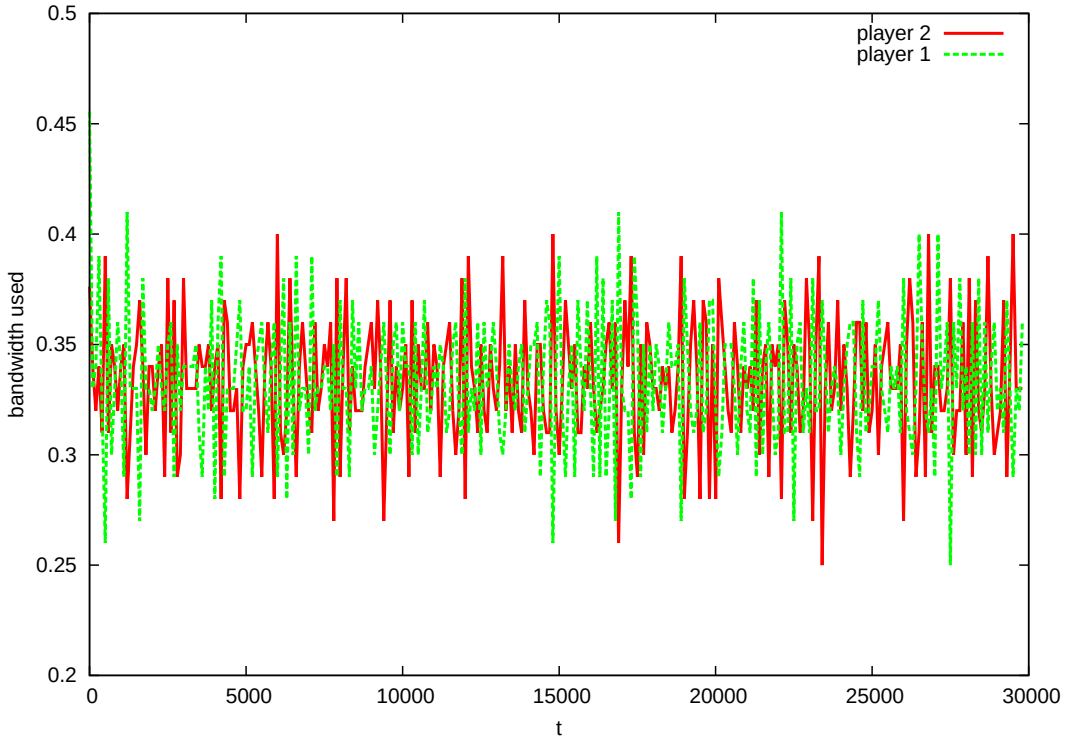


Figure 3.14: Simulation results for $intervalLength = 1$ and plotting interval of 100 time steps

period of the oscillation around the fair solution decreases. This is shown in Figure 3.16, where we used the same plotting interval length as in Figure 3.7.

In conclusion, as long as the *load*-granularity is big enough, we can tradeoff computation time with convergence time and accuracy.

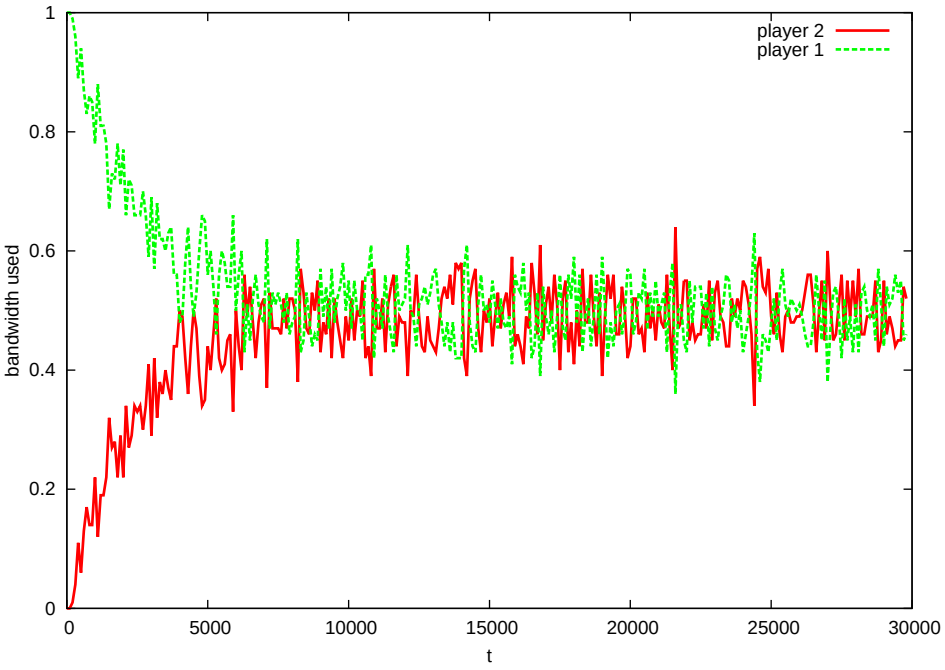


Figure 3.15: Simulation results of the penalty learning algorithm for $l_{mi} = 100$

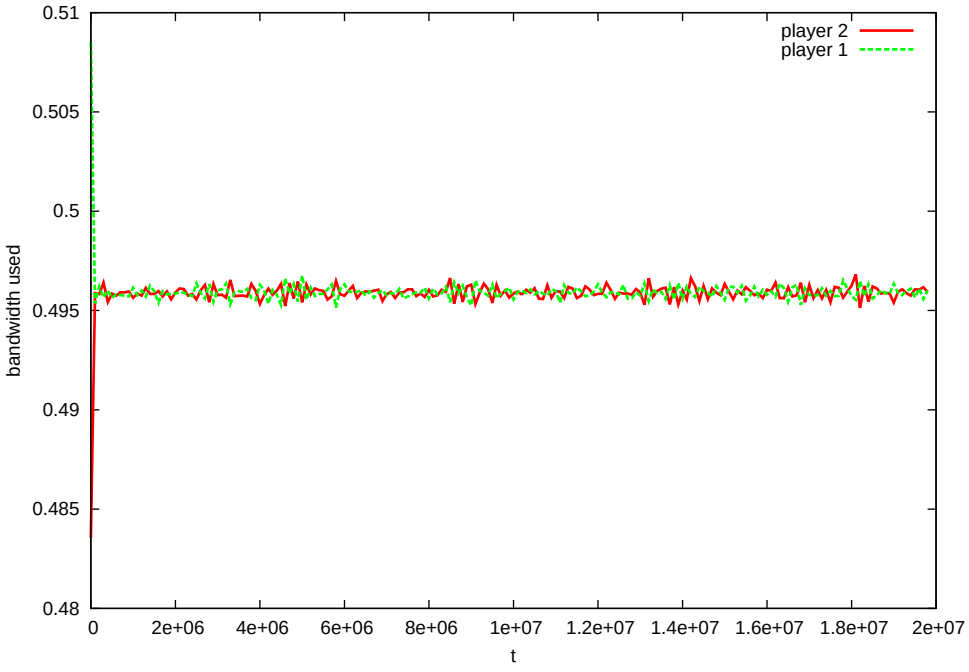


Figure 3.16: Simulation results of the penalty learning algorithm for $l_{mi} = 100$ and plotting interval of 100000 rounds

3.3.2.4 Analysis of ϵ

In the following, we will analyze the influence of the size of ϵ. It is desirable to use as much available bandwidth as possible, therefore ϵ should be reduced to a minimum. In order to evaluate the behavior, the basic setup from Section 3.3.2 is simulated with different ϵ similar to the experiments in Section 3.3.2.1. The results in Figure 3.17 show that no measurable difference in convergence time or accuracy can be witnessed. From this result, we conclude that ϵ should be set to smallest value possible for the implementation.

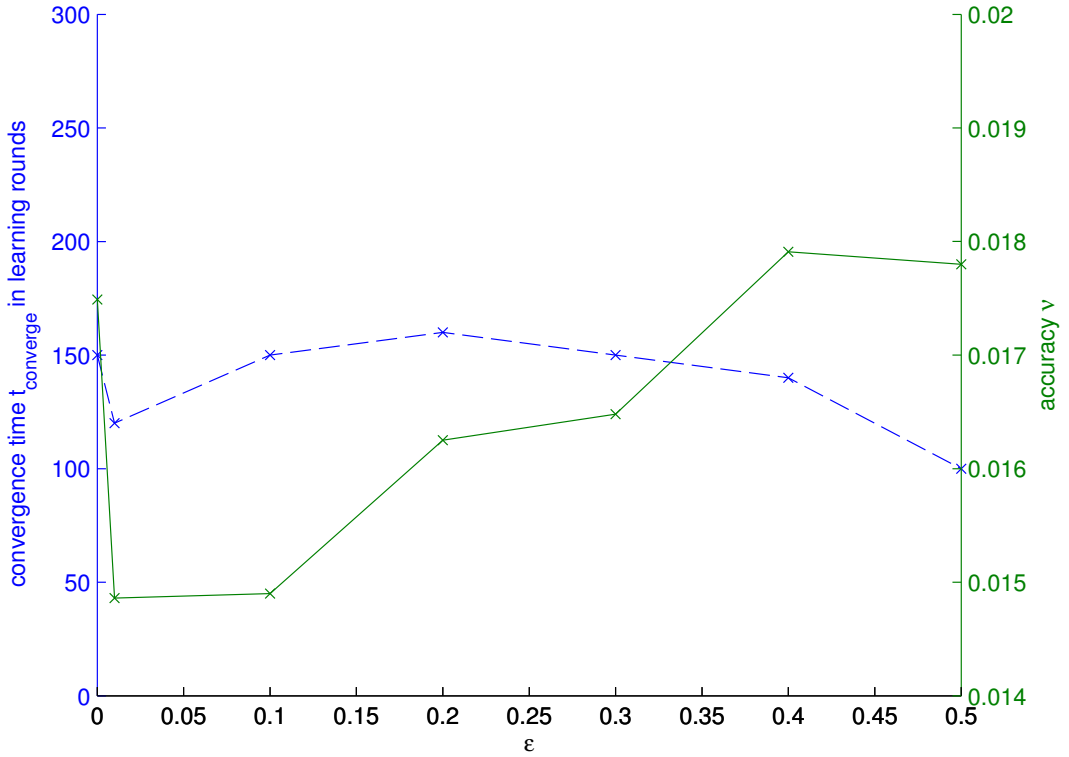


Figure 3.17: Simulation results of the penalty learning algorithm for $l_{mi} = 100000$ and $\eta = 0.02$

3.3.2.5 Summary

As introduced in Section 3.3.2, we have five degrees of freedom for the simulation. The number of streams, i.e., players, is given by the application and only influences the convergence time at the beginning. The remaining four can be chosen by the designer. If the initial number of players is known the starting probability p_i^{start} should be set according to Equation (3.3) with $b = 1$. In this case, the convergence time at the beginning is reduced to 0. ϵ should be set to

the smallest value possible by the implementation, in order to enable the use of the full bandwidth. Decreasing l_{mi} increases the convergence speed, but it also means that the sending probabilities need to be recalculated more often and therefore more computation has to be done. Choosing the learning rate η means trading off between convergence time and accuracy. The bigger steps we take to reach the converged state the bigger are the steps around the desired bandwidth distribution.

To conclude the experiments, no matter how the parameters are set the system converges to a fair bandwidth distribution. Only convergence time, accuracy, used bandwidth, and computation time are traded off by the parameters as summarized in Table 3.2.

3.3.3 Behavior in Mixed Traffic

Assumption 1 in Section 3.2.1 is valid for nodes where the quality of the application varies with the bandwidth they can use. The message streams they produce are called *bandwidth streams*. An example for such a stream is a video application, where depending on the used bandwidth the frame rate or the resolution changes. However, assuming that all streams are bandwidth constrained doesn't reflect current real communication scenarios [NNH05]. On the contrary, the amount of nodes not being bandwidth constrained is many times higher. Most of the nodes belong to the class of response time sensitive as defined in Section 2.1, also called real-time streams. They have different requirements and therefore, need to be scheduled with different methods. In order to allow both types of message streams to be present in the system, we will show that the behavior of the PLA is independent of the scheduling method of the real-time streams.

The advantage of priority-based buses such as CAN is that the scheduling of messages streams with low priorities doesn't influence the streams with high priorities. We use this mechanism to enable a mixed real-time/bandwidth constrained scheduling by assigning the lowest priorities to the bandwidth con-

With increasing:	#players	η	l_{mi}	ϵ
Accuracy	-	less	-	-
Convergence time	larger	less	larger	-
Computation	-*	-	decreased	-
Used bandwidth	-	-	-	less

Table 3.2: Influence of the parameters. - represents no influence. *: The computation per player doesn't change, however the amount of computation of the system increases linear with the number of players.

strained streams. This ensures an undisturbed communication of the real-time streams, while the bandwidth constrained streams share the remaining bandwidth. We can even prove that the bandwidth sharing is not influenced by high priority streams.

Theorem 3.6. *Given a priority-based Medium Access Game with a set of players K playing a fair strategy according to Equation* (3.3)*. Any number of players with higher priority can be added while the players K will share the remaining bandwidth fair.*

Proof. According to Definition 3.3, the utilities of the k players in K are

$$F_1(\mathbf{p}) = F_2(\mathbf{p}) = ... = F_k(\mathbf{p}). \tag{3.5}$$

According to (3.1), it follows

$$p_1 \prod_{j=2}^{k}(1-p_j) = p_2 \prod_{j=3}^{k}(1-p_j) = ... = p_k. \tag{3.6}$$

Adding k_{add} players with higher probability results in utilities for each player i in K

$$F_i(\mathbf{p}) = p_i \prod_{j=i+1}^{k+k_{add}} (1-p_j) = p_i \prod_{j=i+1}^{k} (1-p_j) \prod_{j=k+1}^{k_{add}} (1-p_j). \tag{3.7}$$

By multiplying (3.6) with $\prod_{j=k+1}^{k_{add}}(1-p_j)$, we get

$$\begin{aligned} p_1 \prod_{j=2}^{k}(1-p_j) \prod_{j=k+1}^{k_{add}} (1-p_j) \quad &= \quad p_2 \prod_{j=3}^{k}(1-p_j) \prod_{j=k+1}^{k_{add}} (1-p_j) = \\ ... = p_k \prod_{j=k+1}^{k_{add}} (1-p_j). \end{aligned} \tag{3.8}$$

Comparing Equations (3.7) and (3.8) shows the utilities of all players are still the same and therefore shared fair. □

In the following, we evaluate the theoretical findings by means of simulation. The real-time streams are presented by a player that has higher priority than the bandwidth constrained players and is sending periodically. In Figure 3.18, real-time streams are represented by player 1 sending with a period of 5 rounds and then with a period of 2 rounds. It is simulated with two bandwidth constrained streams. The two bandwidth constrained streams share the remaining bandwidth correctly as shown in Figure 3.18b. Only at start, player 2 needs to converge from a sending probability of 1 to 0.5. If the real-time streams change

their bandwidth use as simulated at round 200,000 the bandwidth constrained streams instantly adapt, because the sending probabilities don't have to change. The sending probabilities are shown in Figure 3.18a. The experiments reflect the statement of Theorem 3.6. We can therefore conclude that our self-organizing mechanism can co-exist with scheduling approaches for other quality of service requirements.

3.4 Periodic Access Method

In this section, we propose to use a sending period instead of a sending probability to control the access to the medium. This has two reasons: First, many applications, especially sensor data collection, typically use the periodic access scheme such as described in [BLCA02], thus limiting the usability of our probabilistic access scheme. Second, the deterministic behavior of the period access scheme improves the convergence time and accuracy.

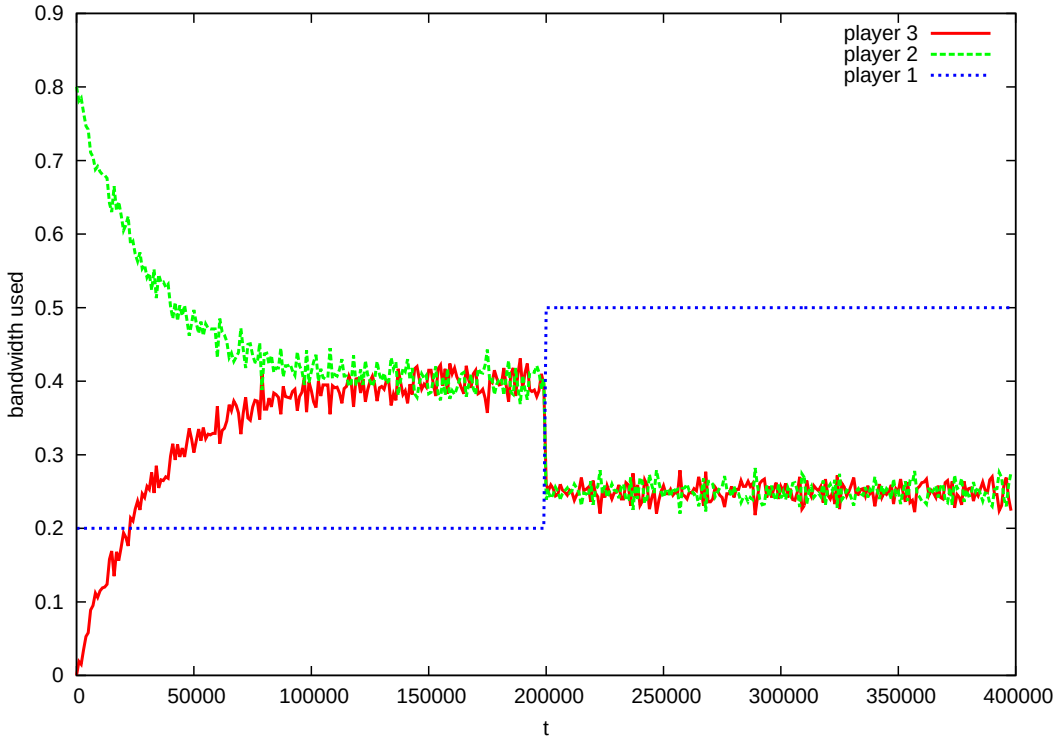
In the following, first, the algorithm is presented. Then, it is analyzed by software experiments. Finally, a procedure is introduced that guides the designer in setting the parameters of the algorithm.

3.4.1 Period Penalty Learning Algorithm (PPLA)

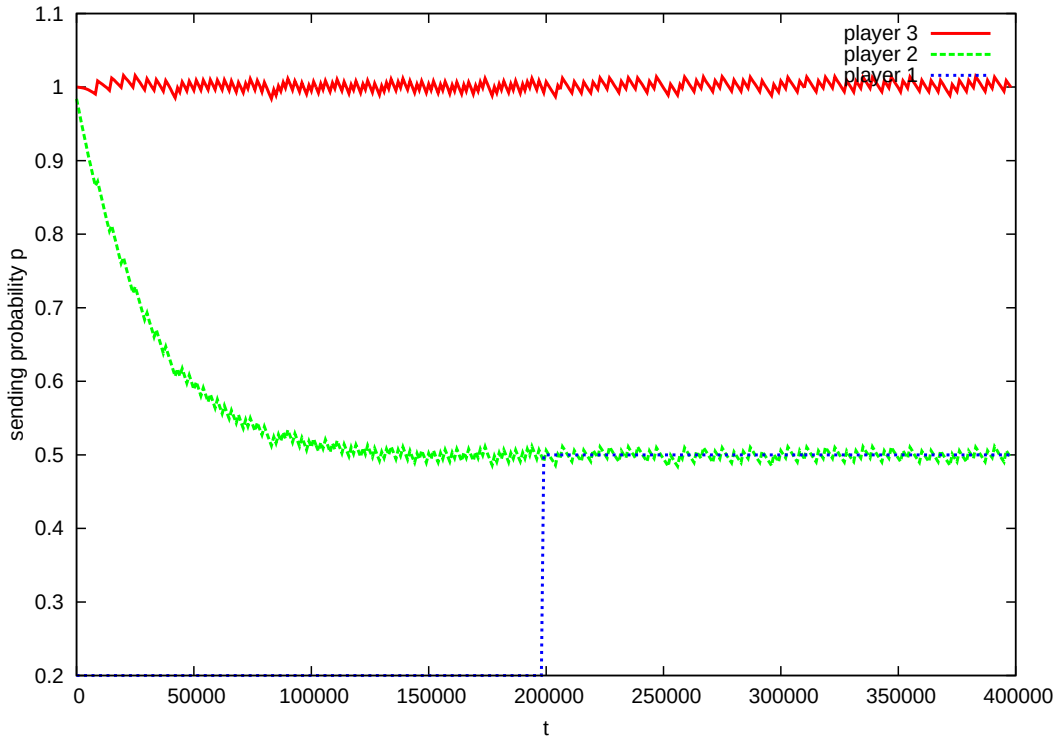
This algorithm is called Period Penalty Learning Algorithm (PPLA), because it is derived from the PLA described in the previous section. However, in this algorithm the sending period T_i determines the number of rounds player i waits until he tries to send. The number of rounds doesn't need to be integer. For example, if the period is 2.5 the player will try after 3, 2, 3, 2, ... rounds. Another difference to the probability access scheme is, that if the player is not granted access when trying to send, he remembers this message and tries to send it in the next round. This could of course lead to a buffer overflow. To prevent this, these stored messages are cleared after one learning step.

As the PPLA is almost similar to the PLA, we describe the operation of the PPLA by adapting Algorithm 3.1 slightly. Instead of producing a negative Δ in line 4, the minus is removed. In line 6, a minus is added. In summary, the if-clause is changed as described in Algorithm 3.2. This adaptation stems from the fact that the bandwidth demand increases inversely to the period of the stream.

Using the period access scheme, the behavior of the players is state-dependent. Therefore, we can use our game theoretic model only to describe the problem, but it is very difficult to analyze and prove it. Nevertheless, the following positive aspects suggest the use of the period access scheme. The main positive aspect is that the period access scheme is deterministic, which means, each



(a)



(b)

Figure 3.18: Simulation results for PLA with player 1 representing real-time traffic, $l_{mi} = 1000$ and $\eta = 0.02$

Algorithm 3.2 Adjustment of T_i for PPLA during one learning step

1: $load = \frac{m_{sent}}{l_{mi}}$;
2: $success = \frac{nSuccess}{l_{mi}}$;
3: **if** $(load > 1 - \epsilon)$ **then**
4: $\quad \Delta = success$
5: **else**
6: $\quad \Delta = -(1 - success)$
7: **end if**
8: $T_i^{(t+1)} = T_i^{(t)} + \eta \cdot \Delta$;

learning step played with the same periods will return exactly the same reward to all players. Therefore, wrong learning decisions because of the actual access deviating from the chosen one cannot happen. This determinism improves convergence speed and accuracy, and also makes it possible to calculate upper bounds for player strategies beforehand. Another reason to use the periodic access scheme is the common usage in control applications. In the following, we give a detailed analysis of the algorithm's parameters and discuss how to tune them to optimize the algorithm's behavior.

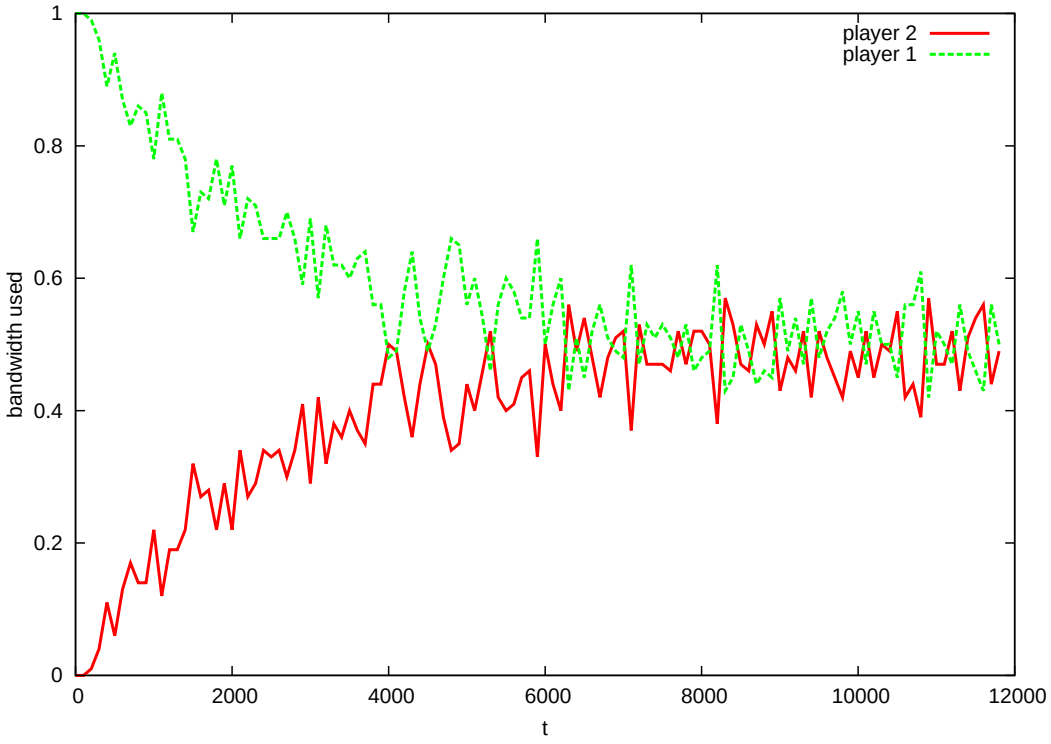
3.4.2 Experimental Evaluation

Equal to the PLA in Section 3.3.1, the PPLA has 5 degrees of freedom. The experimental analysis shows, as expected, that the parameters have the same tendencies as the PLA. Consequently, Table 3.2 is also valid for PPLA. The difference to the PLA is the absolute value of convergence speed and accuracy. In the following, the convergence speed of both approaches are compared. To do so, both algorithms are run with $T_i^{start} = 1$ or $p_i^{start} = 1$, $k = 2$, $\epsilon = 10^{-15}$, $l_{mi} = 100$. η was by experiments adjusted to result in an accuracy of 0.1. This means PLA is run with $\eta = 0.02$ and PPLA with $\eta = 0.5$. The results are shown in Figure 3.19. The convergence speed of PPLA is at least one order of magnitude better. The influence of k and η on the accuracy is analyzed in Section 3.4.4.

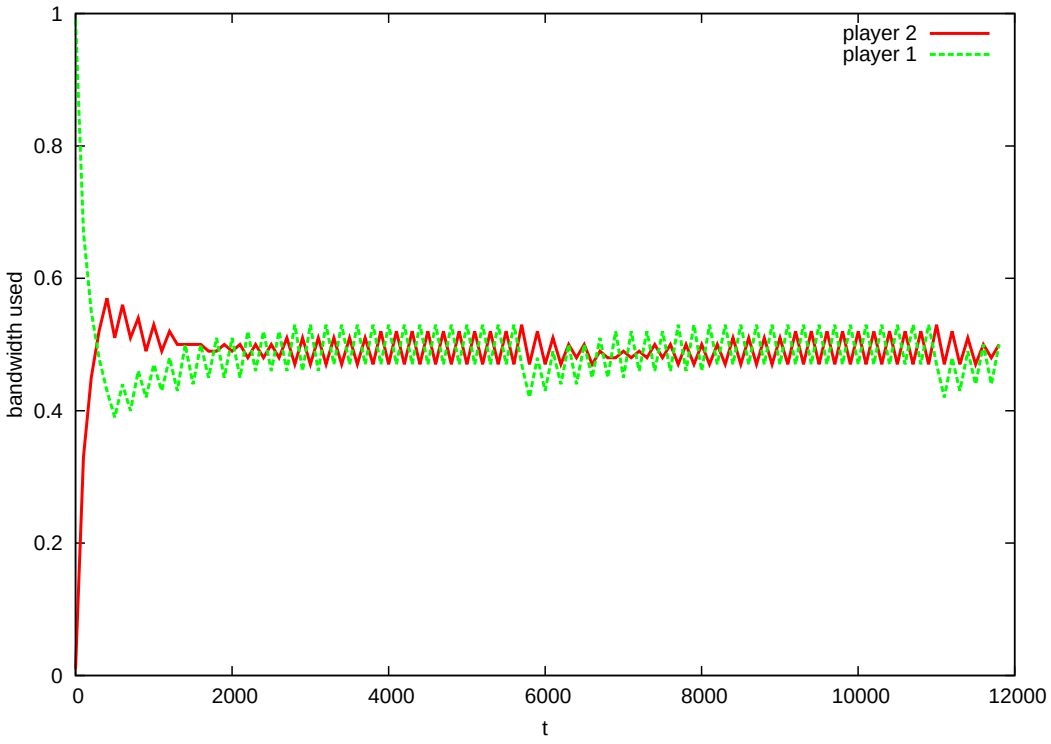
3.4.3 Accuracy

In order to calculate the accuracy at the converged state for the PPLA, we make the following assumptions on the parameters described in Section 3.3.2: $k \geq 2$, $\alpha =]0, 0.5[$ and $\eta =]0, 1[$.

First, we assume that the players play fair at the converged state. This means, the period T_i chosen by all players i is $T_i = T_{fair} = \frac{1}{\alpha}$ and $success = \alpha$. According to Algorithm 3.2, Δ can either be $success = \alpha$ or $-(1 - success) =$



(a) PLA with $\eta = 0.02$



(b) PPLA with $\eta = 0.5$

Figure 3.19: Simulation results for PLA and PPLA with $l_{mi} = 100$

$-(1-\alpha)$. Since $\alpha < 0.5$, it follows that $|\alpha| < |-(1-\alpha)|$. Therefore, the maximum of the absolute value of Δ is $1-\alpha$. From this follows, the maximum step size $\delta T = \eta \cdot (1-\alpha)$. The minimum period a player can reach in one step is $T_{min} = T_{fair} - \delta T = \frac{1}{\alpha} - \eta \cdot (1-\alpha)$. The maximum bandwidth that can be reached is the reciprocal $\alpha_{max} = \frac{1}{T_{min}} = \frac{1}{\frac{1}{\alpha} - \eta \cdot (1-\alpha)}$. Consequently, the minimal accuracy when all players play the fair bandwidth is

$$\mu_{i>1} = \alpha_{max} - \alpha = \frac{1}{\frac{1}{\alpha} - \eta \cdot (1-\alpha)} - \alpha. \tag{3.9}$$

Equation (3.9) is valid for $i < k$. As the example in Figure 3.20 shows, player 5 is not playing the fair bandwidth. This doesn't influence the bandwidth the players get, because player k has the lowest priority and therefore just gets the rest. However, this also means if the other players lower their used bandwidth by increasing their periods, player k uses this free bandwidth for one round. This decreases the accuracy of player k by the sum of the accuracies of the other players. As the accuracy of player k is the worst, the overall minimal accuracy is the minimal accuracy of player k, that is

$$\mu_1 = (\alpha_{max} - \alpha) \cdot (k-1). \tag{3.10}$$

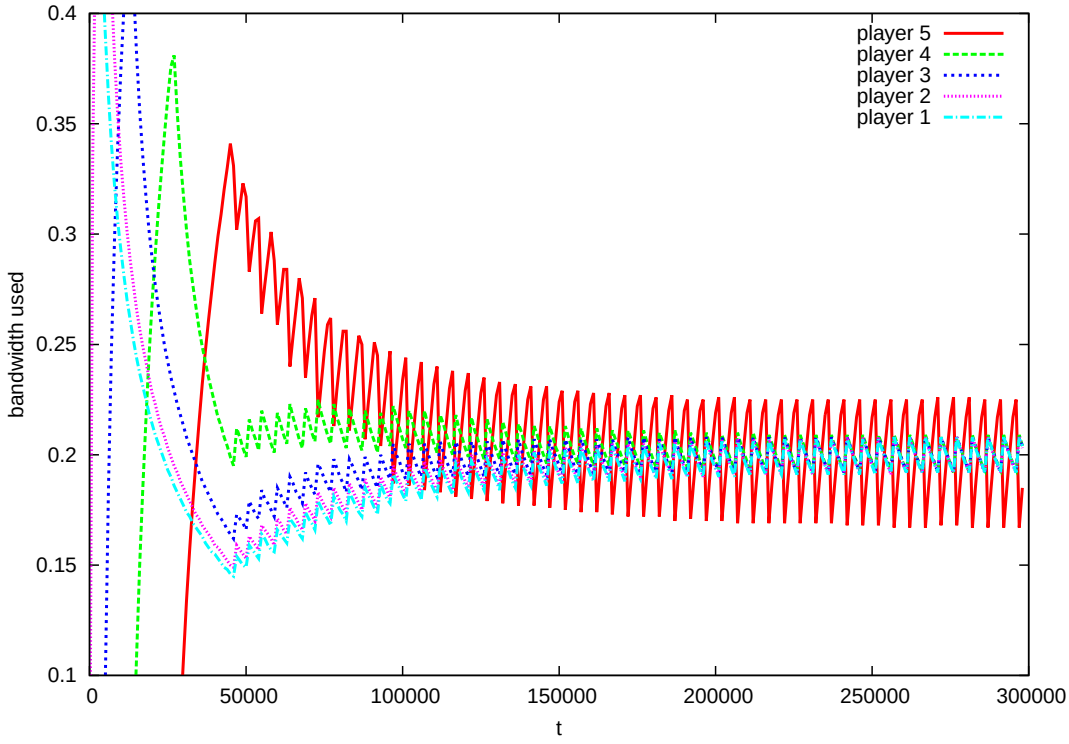
In Figure 3.21, the minimal accuracy is plotted for different η over α. The plot shows that the worst accuracy is at $\alpha = \frac{1}{3}$, i.e., $k = 3$. Inserting these values in Equation (3.10), the worst case accuracy is $(\frac{1}{\frac{1}{\frac{1}{3}} - \eta \cdot (1-\frac{1}{3})} - \frac{1}{3}) \cdot 3 - 1 = \frac{\frac{4}{9}\eta}{3-\frac{2}{3}\eta}$.

In addition, the accuracy is constrained by the length of the monitoring interval l_{mi}, because the minimal change a player can perform is in the granularity of rounds, i.e., messages. Consequently, the minimal error is $\frac{1}{l_{mi}}$. In summary the accuracy is:

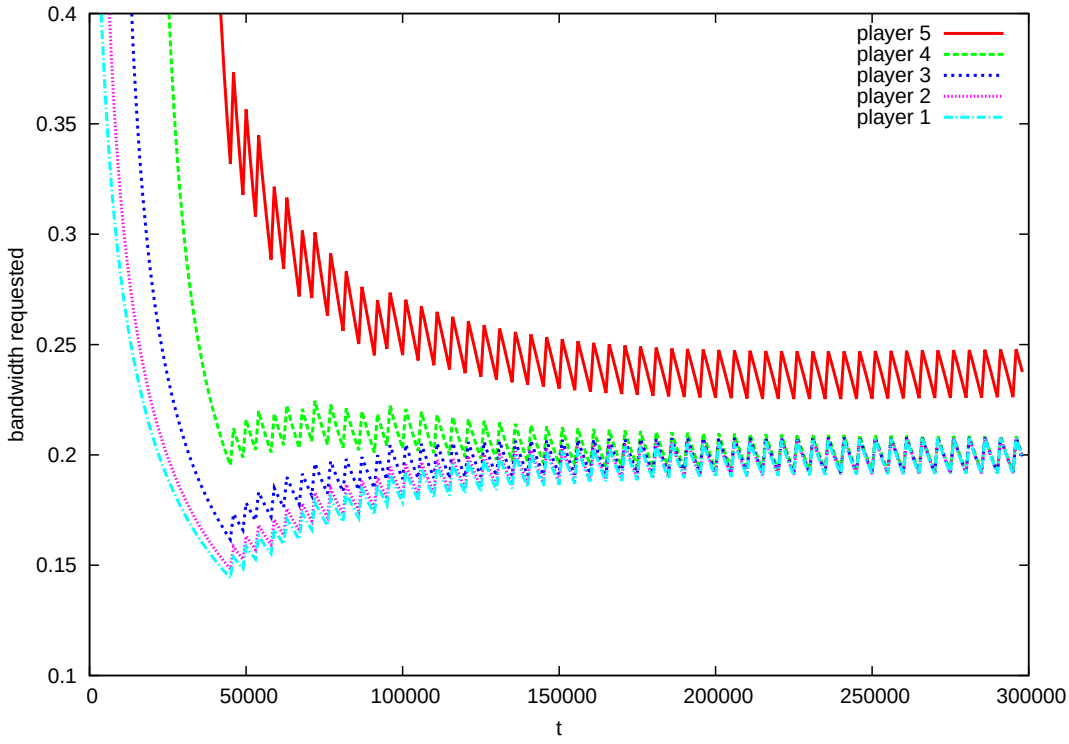
$$\mu = max \left\{ \frac{\frac{4}{9}\eta}{3 - \frac{2}{3}\eta}, \frac{1}{l_{mi}} \right\} \tag{3.11}$$

3.4.4 Setting of the Parameters

Self-organizing systems, especially when they use learning algorithms, often have the problem that the adjustment of the learning parameters is difficult. The performance of many learners depends on the correct setting of parameters [LD06]. The difficulty is that most often it is not understood what effect the parameters have. This problem is amplified by the fact that there are too many parameters to evaluate them exhaustively. We want to counteract this problem for our algorithms by proposing a standard procedure to set the parameters of the PPLA.



(a) Bandwidth the player gets



(b) Bandwidth the player would get according to the set period

Figure 3.20: Simulation results for PPLA with $l_{mi} = 1000$, $k = 5$ and $\eta = 0.5$

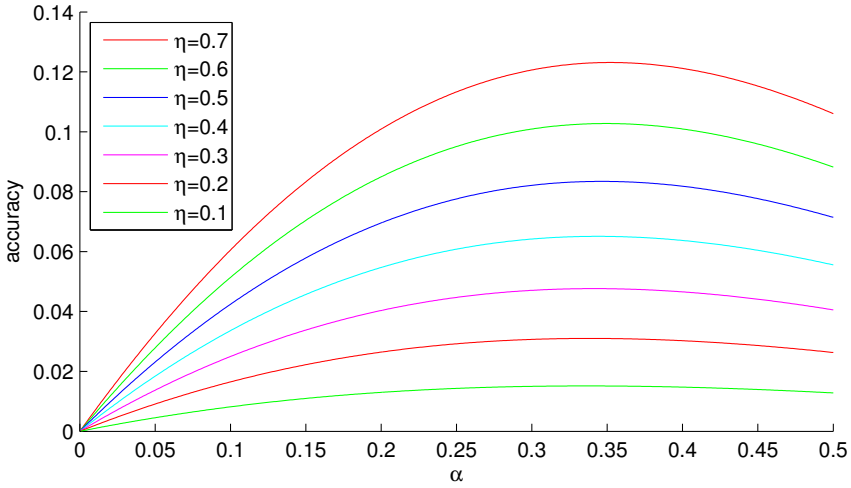


Figure 3.21: The minimal accuracy over α for different η

The procedure consists of two steps: First, the designer has to choose an interval and an accuracy μ for this interval. These values are application dependent. For example, for a video stream it would make sense to set the interval to the period that corresponds to the frame rate. The accuracy would depend on how much difference the quality of experience is when the used bandwidth is different. The only condition the designer has to take care of is that the accuracy is greater than the reciprocal of the interval. With the assumption $\mu > \frac{1}{l_{mi}}$, Equation (3.11) simplifies to

$$\mu = \frac{\frac{4}{9}\eta}{3 - \frac{2}{3}\eta}. \tag{3.12}$$

Taking this into account η can be calculated by

$$\eta = \frac{9 \cdot \mu}{\frac{4}{3} + 2 \cdot \mu}. \tag{3.13}$$

In the second step, the only undecided parameter is the length of the monitoring interval l_{mi}. The designer can set it to the chosen interval to have the desired accuracy with minimal computation overhead. However, to increase the convergence speed l_{mi} can be further reduced, providing a trade-off between convergence speed and computation overhead. An additional positive effect of decreasing l_{mi} is that also the accuracy improves.

3.5 Experimental Evaluation on CAN

Until now, the simulations were based on the game theoretic model. The streams competed synchronous round by round for the access to the communication medium. In the following, we take the next step towards a real setup and test our algorithms on a CAN simulation.

3.5.1 Simulation Setup

We developed and used our own CAN bus simulator, because other known available simulators, such as for example RTaW-Sim [RS09], are not able to deal with all scenarios we apply and cannot extract the required figures for the properties explored. Our simulator is discrete-event driven with the simulation step equal to one CAN bit and assumes the error free case. We follow an object-oriented approach using Java as the design language. The main flow of the simulation is depicted in Figure 3.22. The flow consist of three steps: Loading of the scenario, simulating each bit for a fixed number of time steps and then storing the gathered results in external files.

The first step is to initialize the simulation by specifying the amount and type of streams. For each message stream it contains a name, a priority, a period, a length, and an offset. In this section only bandwidth streams are used that transmit periodically with a fixed offset and a maximal message length of eight byte to reduce the overhead. Most of the simulation time is spent in the second step. Therefore, we will detail the functional principle of this step. Three different objects are mainly responsible for the simulation: *Sim*, *Bus,* and *Stream*. The Sim object contains the main method and controls the course of action. The Bus object stores the information of the current bus state, while

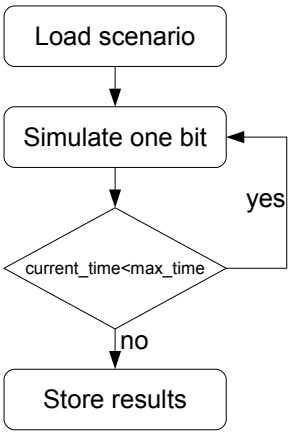


Figure 3.22: Main flow of the simulation.

the Stream object stores the information only for its stream. Two example operating sequences are shown in Figure 3.23. In the first sequence, the bus is free, which is indicated by the Bus. At the beginning, the Sim asks all streams whether they have something to send. Next, the streams that have something to send are notified whether they are sent or not. The decision which one will be sent is based on the priority of the streams. Finally, the Bus and all Streams are informed which Stream sent a message. The Bus uses this information to calculate how long the message will occupy the bus. The length of each message is calculated by assuming worst case bit-stuffing (see Section 2.2.2.2). The second sequence shows the case where the bus is busy.

Because the Stream object is the heart of the simulation and because the adaptation algorithm is integrated into the Stream's behavior, this object is

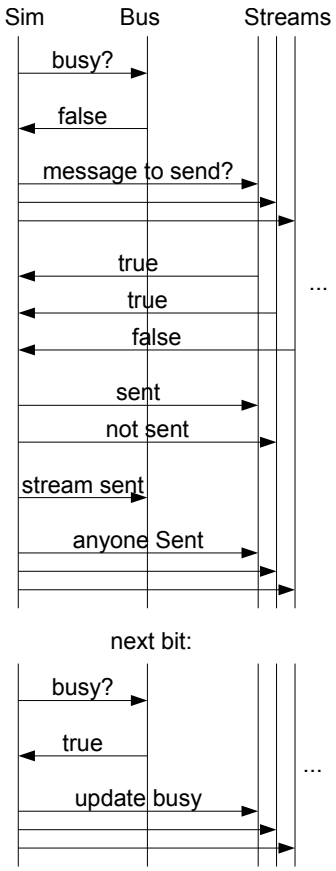


Figure 3.23: Two example operating sequences of the main simulation engine.

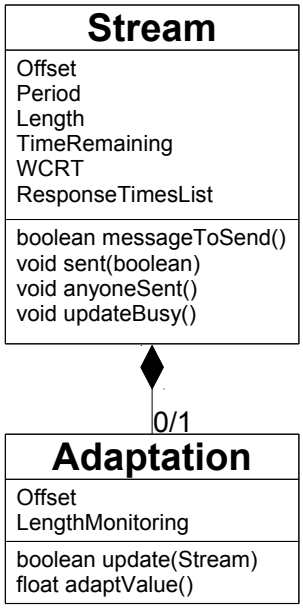


Figure 3.24: Class diagram of the Stream class.

described in more detail. Figure 3.24 shows the class diagram of the Stream. Besides the static parameter of a message stream, the Stream stores the number of time steps until the next message should be transmitted, the worst case response time, and a list of the last response times. For bandwidth streams the response times are not relevant, but for the evaluation of soft real-time streams in Chapter 4 it is necessary to log the response times. If the stream is adapting, it has an Adaptation object. It needs to be updated every bit and the calculated adaptation value needs to be requested when the update function returns true, indicating that the end of a monitoring interval has been reached. In the case of the PPLA the period of the stream is adapted. Hence, the adaptation value is the period for the next monitoring interval.

3.5.2 Results

The main difference of using the PPLA on the CAN bus compared to the game theoretic model is that the arbitration phases are not synchronized and therefore one round can not be correlated to one message transmission. For the algorithm to work it is important that for each round it is clear who won the arbitration. For this reason, we propose to use the transmission time of one CAN bit as one round. This has the consequence that the transmission of one message takes several rounds. This means that the monitoring interval and the total simulation time have to be multiplied by the length of the message transmission. The length of a message with maximal payload of 8 bytes varies due to bit stuffing between 111 and 136 bits. In the following, an estimate of 100 bits is used. Furthermore, the adaptation of the period in line 8 of Algorithm 3.2 on page 54 needs to be scaled by the message transmission length. In the first experiment, the monitoring interval is synchronized, i.e., the offset of the monitoring intervals is zero. The results in Figure 3.25 and Figure 3.26 show that the behavior is almost identical to the game theoretic model in Figure 3.19b and Figure 3.20. With these results, it is now possible to provide realistic numbers of the time of convergence. One time step in the simulation corresponds to the time of the transmission of one CAN bit. Depending on the used bit rate, the bit time is typically either 8μs or 2μs for 125 kbit/s and 500 kbit/s respectively. This means for example for the setup in Figure 3.25 where the converged state is reach after $1 \cdot 10^5$ simulation time steps, the convergence time is 800 ms or 200 ms.

On a real CAN, the nodes are not synchronized and therefore the monitoring intervals are offset randomly. In Figure 3.27 the result for this effect is simulated with the same settings as Figure 3.20 on page 57. This experiment shows that the used bandwidth is slightly randomly disturbed, but the convergence time and the accuracy are almost similar.

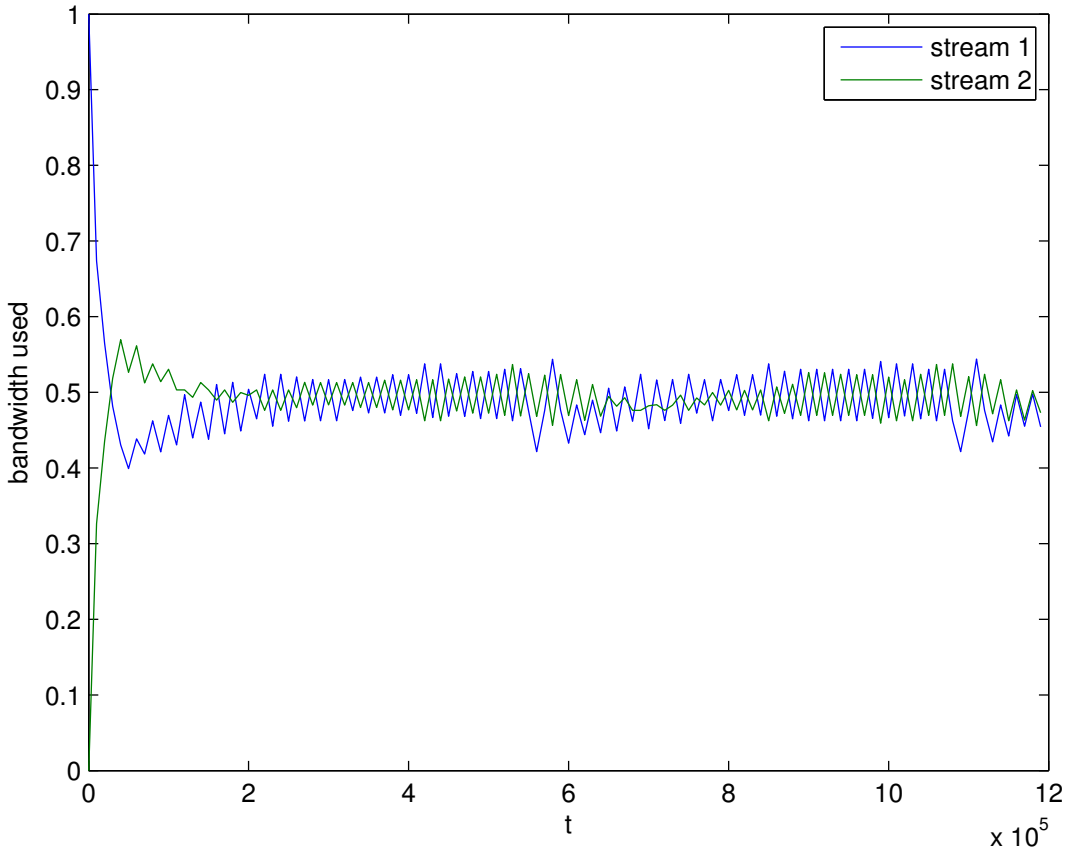
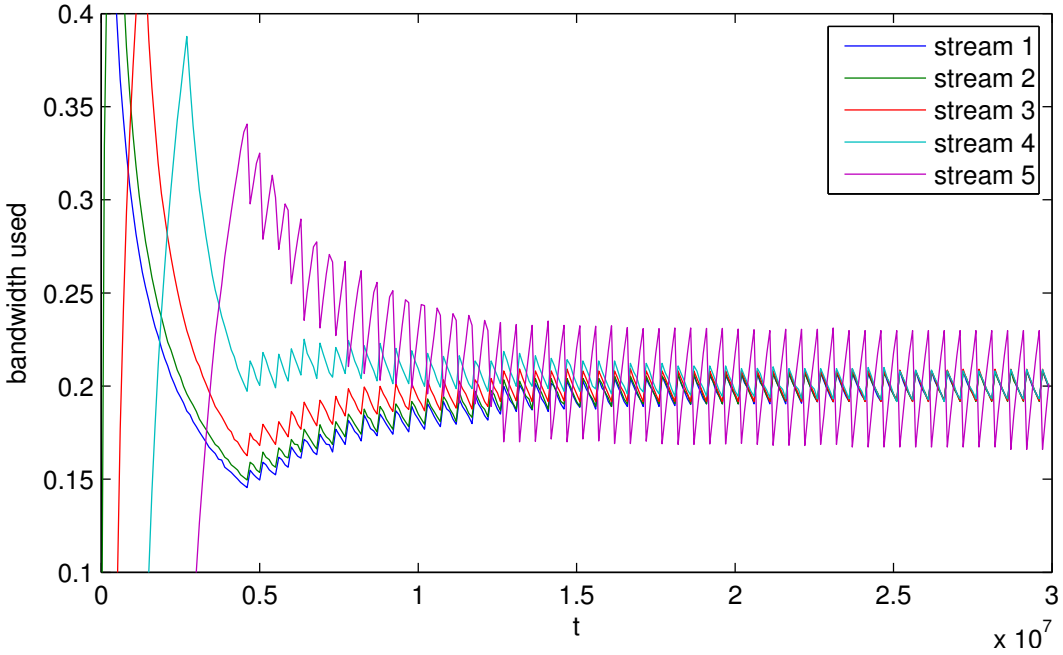
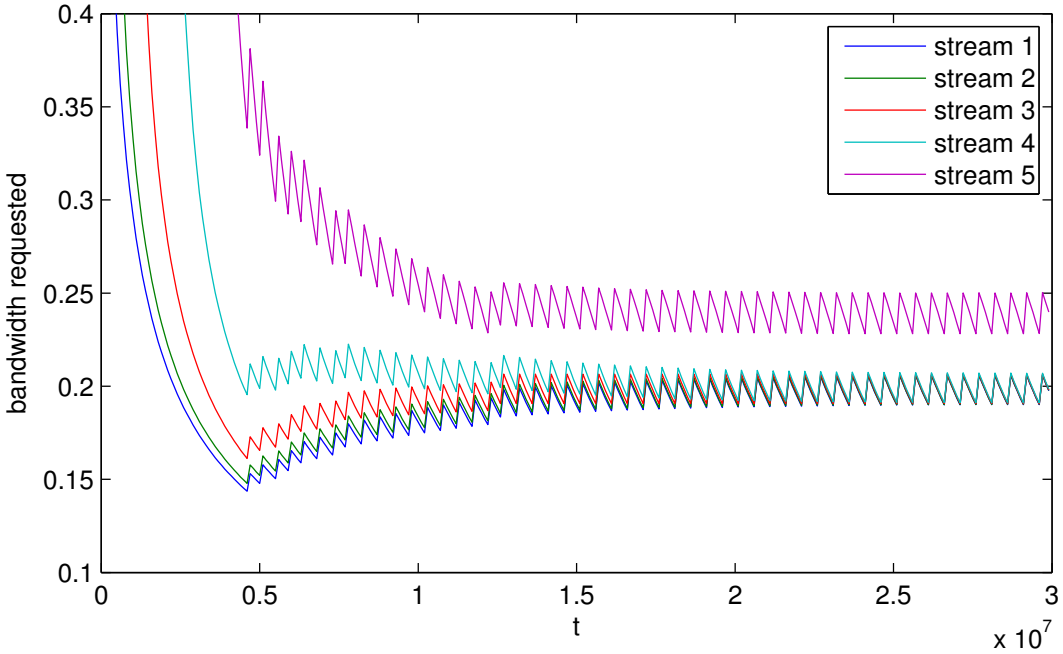


Figure 3.25: PPLA with $l_{mi} = 10000$ and $\eta = 0.5$ run on CAN simulation.



(a) Bandwidth the stream gets



(b) Bandwidth the stream would get according to the set period

Figure 3.26: Simulation results for PPLA with $l_{mi} = 100000$, $k = 5$ and $\eta = 0.5$

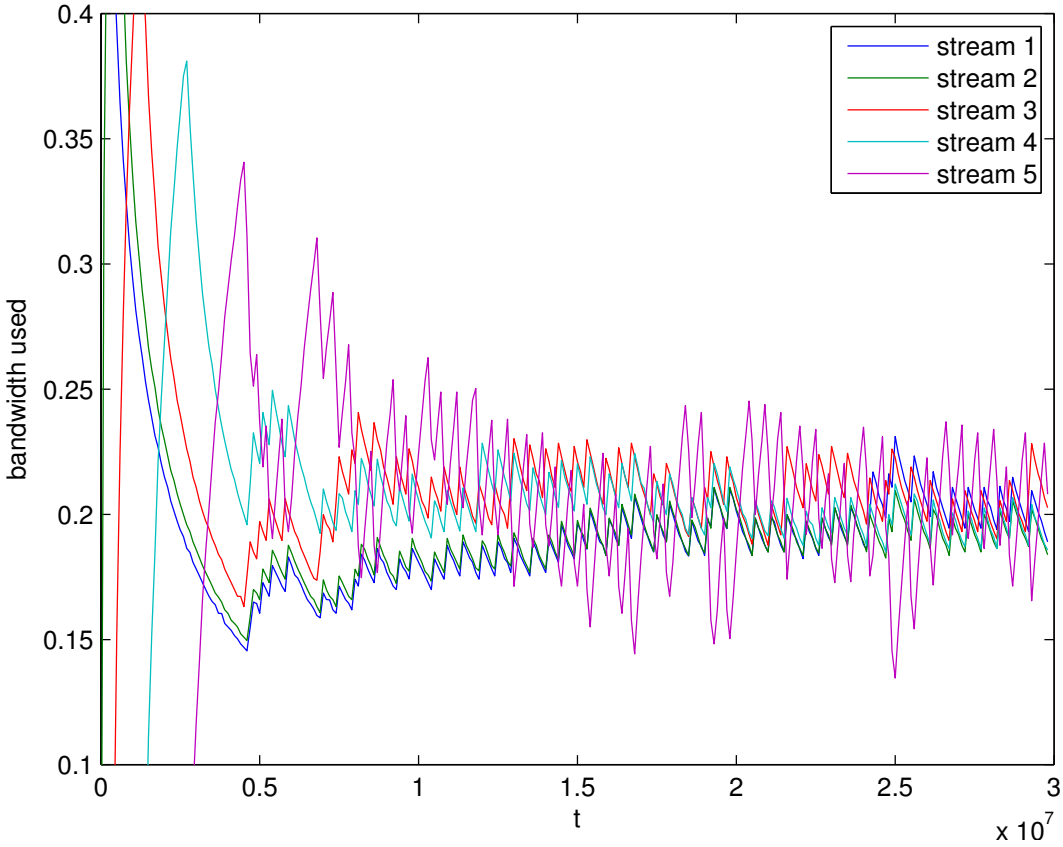


Figure 3.27: PPLA with $l_{mi} = 100000$, $k = 5$ and $\eta = 0.5$ run on CAN simulation with random offset of the monitoring interval.

4

Dynamic Adaptation of Offsets to Reduce Response Times

Another important class of message streams is the class of soft real-time streams (cf. 2.1.2). These message streams are used to connect sensor-controller-actuator chains which represent the majority of applications in DES. In contrast to bandwidth streams of the previous chapter, the challenge here lies in the response time of the messages. As long as the load of the priority-based bus is below a safe limit, the arbitration mechanism itself ensures sufficiently short response times. According to Navet and Perrault [NP12], this safe limit is between 35 and 40% in practical applications. If the workload increases beyond this limit, the concurrent release of messages can lead to unacceptable delays for the messages with lower priorities. In the following chapter, we try to solve this problem by scheduling the messages during run-time such that concurrent releases are avoided and therefore the response times reduced.

The chapter is organized as follows. In Section 4.1 the problem is defined and placed into the context of the related work. In Section 4.2, we introduce a rating function that allows the comparison of different schedules by a single value. Section 4.3 presents the algorithm to dynamically adapt offsets to reduce response times and compares it qualitatively to comparable approaches. In Section 4.4, CAN simulations are used to analyze our approach quantitatively. Then, the dynamic offset adaptation is extended in two ways: (1) In Section 4.5, the necessary changes to support multi-segment systems are presented and (2) in Section 4.6 an initialization mechanism is introduced.

4.1 Related Work and Problem Definition

The trend towards adding more nodes to the distributed embedded and control systems applications faces the limited bandwidth and message response times as the major obstacles. An obvious solution to this problem is to try using faster buses such as FlexRay [FC09] or Ethernet [Fel05]. However, the introduction of new bus technology is linked with a huge cost, because it requires the change of not only the bus infrastructure, but also modifications of both hardware and software on the ECUs. Furthermore, the faster communication bus protocols use different arbitration methods. For example, the arbitration of FlexRay offers much less flexibility as described in Section 2.3.1. On the dynamic segment, the response times are less predictable and in average larger than purely priority-based approaches. The static segment offers fixed time-triggered access that wastes bandwidth if the reserved slots are not used. Comparison of time-triggered and event triggered approaches has been subject of many discussions [Alb04] and there is a sound argument of the advantage of using time-triggered approaches for systems with purely periodic message release.

However, as many existing and future systems still rely on use of traditional buses, in our case CAN, the focus of many researchers has been how to better utilize CAN and effectively extend its applicability and life. Besides retaining legacy systems and huge investment into various kinds of nodes and applications, CAN still offers its original advantages of simple implementation, short response times, and ease of change of the designed system. In some applications, e.g., automotive, the applicability of CAN has been extended by using several bus segments, which accommodate smaller number of nodes, interconnected through a gateway [SBF06]. However, this approach results in increased response times, because the gateway itself introduces additional delays. Moreover, analyzing the timing behavior of such systems is more difficult, because these delays have to be estimated and taken into account in system design.

There has been a significant body of research on how to allow a time-triggered access on CAN by using new protocols on top of CAN. In that case, a single node organizes the scheduling of the messages, e.g., the FTT-CAN Protocol [APF02] or the server-based communication [NNH05]. Even though the guaranteed response times are improved, the disadvantages of these approaches are that the scheduling node (master) needs to send extra scheduling messages and all other nodes (slaves) need to register their transmission. This causes additional traffic over the bus that further reduces the already small bandwidth. Another disadvantage is that all nodes need to follow this protocol, making incremental changes in system design and integration difficult.

Several approaches try to optimize the scheduling of messages on CAN by adjusting the priority of the messages. One way is to divide the message streams into different categories and schedule them with known scheduling techniques

such as earliest deadline first [ZS95, DN00]. Another way is to use fuzzy logic to select the priority [BHWY05]. The different priorities are established by using several bits from the identifier. This has the disadvantage that the number of available identifiers is reduced from 2048 to 32 and 128, respectively. Already in current automotive applications more than 128 identifiers are needed, which makes this suggestion infeasible. We assume the priorities are given by the designer according to application specific requisites.

The approach proposed in this thesis addresses the aforementioned disadvantages by using a fully distributed approach to message scheduling without introducing additional communication overhead. Guaranteeing response times is also possible by using an analytical approach and taking advantage of the fact that several message streams are on the same ECU. Fixed offsets between the message streams on one node are assigned by using an offline heuristic algorithm [GCGCF08]. In the remaining text, we refer to it as to *Static Offset Assignment Algorithm (staticOAA)*. The basic idea of the staticOAA is to spread the release times of all messages as far as possible. This is achieved through creation of an array of possible start time slots within the length of the largest message release period for each node. The message streams are ordered according to increasing period. For each stream the following steps are performed:

1. Find the longest interval in which there is no load on the bus,.
2. Set the message release of the stream in the middle of this interval.
3. Update the array with all messages of this stream. According to the resulting schedule, the offsets are assigned for each stream.

The authors have shown that the WCRTs for the streams are reduced compared to the worst case scenario in which all offsets are zero. Assigning offsets to the messages results in better response times, but takes the asynchronous nature of the nodes that communicate over bus only into account by considering worst case assumptions. Even though the asynchronous nature could be eliminated by using a clock synchronization protocol [LA03], the approach does not take into account the dynamics of the system and resulting variations in the bus traffic. Instead, everything is based on a-priori given, static assumptions. In contrast to the staticOAA, the approach proposed in this work is based on monitoring of the current network traffic and adapting message offsets in order to reduce the likelihood of the simultaneous message releases on different nodes.

The approach of dynamically adapting offsets is also used in the domain of WSNs. The distributed algorithm DESYNC [DRPN07] is executed on each wireless node independently. Each node monitors the point in time of the following and the previous message of the other streams. Then, it adapts its offset to schedule the next occurrence in the middle of these two messages. It can be

proven that the message streams are distributed equally over time, if all nodes stick to the protocol. The main limitation of the approach is that only message streams with equal period are allowed that is not the case for the scenarios used here.

According to the system model described in Section 2.1, the message streams in this chapter have following properties. Because our algorithm considers the entire distributed system with all message generating streams, the fact that one or more streams are on the same node can be abstracted and we will assume only one stream per node, which does not affect the generality of the presented results. The occurrence scheme U_i is always periodic and therefore described by a period T_i and an offset O_i. The requirement R_i is soft deadline. This means that it is desirable to keep the response times as short a possible. Additionally, we consider message streams with different lengths l_i. In summary a message stream s is characterized by the tuple (T_i, O_i, l_i).

4.2 Rating of the Schedule

In real-time task scheduling, the quality criterion of a schedule is typically based on a measure of whether or how many deadlines are met. This metric is inadequate for many practical problems because it is not so easy to define a deadline. If we look, for example, at a classical control application, a sensor-controller-actuator chain is triggered periodically and therefore fits to the task model. The problem with control algorithms is that one cannot decide beforehand what maximum delay and jitter the algorithm can tolerate, especially because the delay and jitter may increase over time, and the deadlines cannot be provided in advance.

Another method to evaluate the system performance is to compare WCRTs of all streams as in [GCGCF08]. An example for an automotive scenario is presented in Figure 4.1. The details of scenario ls_50 are provided in Section 4.4.1. The fully asynchronous case assumes that all streams are asynchronous and, therefore, can all occur simultaneously. The second case assumes that the streams on each node are synchronized and the offsets are determined by staticOAA. In contrast to the first two approaches, in the third and fourth approach response times are not guaranteed WCRTs but correspond to maximum and average of response times found in a large number (in this case 1000) of different simulation runs with random uniformly distributed offsets. Here, the assumption that all streams are synchronous is used, i.e., the offsets do not change. This assumption is considered reasonable, as the different clocks drift slowly and the offsets stay unchanged for rather long time intervals.

Comparison of different scheduling approaches typically uses WCRTs of all streams (or tasks). However, this procedure can in some cases not clearly distin-

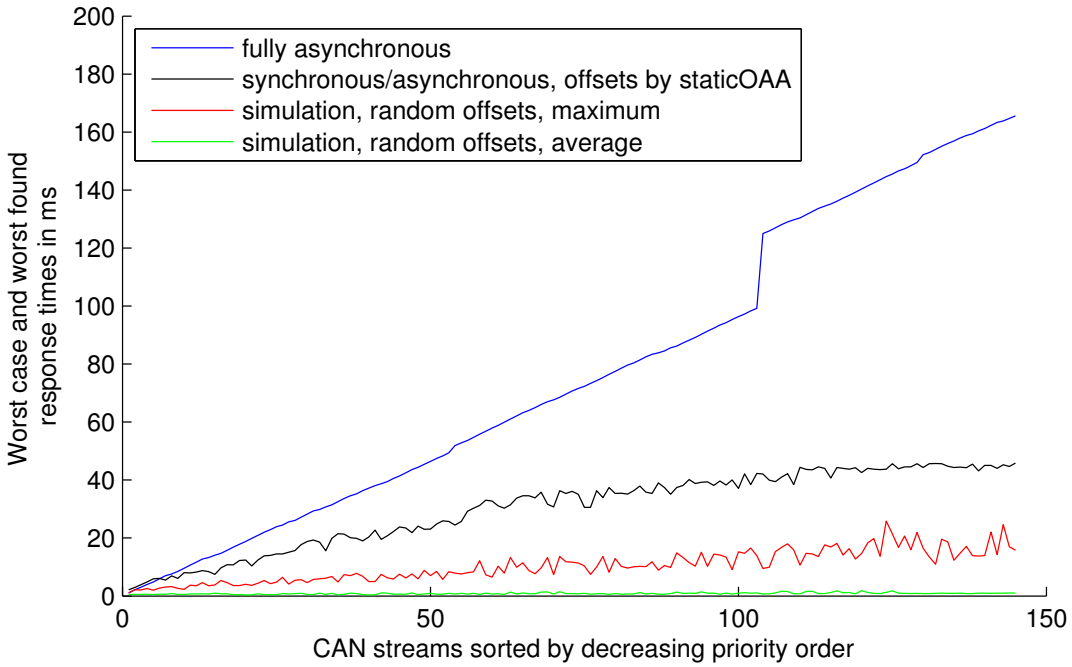


Figure 4.1: Worst case response times and worst observed response times for different synchronism approaches for scenario ls_50.

guish whether one schedule is better than the other. For example, one schedule tends to give advantage to certain groups of streams (e.g., those with low priority) but at the same time being worse for the streams with high priority. In addition, when the response times change over time, it would be good to have a single criterion equally suitable for all types of streams. In this work, we use the rating function called the Average Weighted WCRT (AWW), which enables us to differentiate the quality of different message scheduling schemes:

$$AWW(t_a, t_b) = \frac{\sum_{i=1}^{k} \frac{WCRT_i(t_a,t_b)}{T_i}}{k}, \tag{4.1}$$

where $WCRT_i(t_a, t_b)$ is the measured worst case response time of the stream i that has period T_i in the observed time interval between t_a and t_b and k is the total number of streams generating messages.

The function $AWW(t_a, t_b)$ takes into account that the streams with large periods are more sensitive to large response times, because each WCRT is weighted with its corresponding period. Finally, the sum is divided by the number of streams to get the average rating per stream. This also allows comparison of different scenarios with different numbers of streams.

It would be reasonable to think of additionally weighting the rating by the CAN priorities, but the priorities are already usually taken into account by the

stream periods and, therefore, this would lead to overweighting of messages with low priorities. Furthermore, the WCRTs could be weighted by the designer if any additional knowledge of the importance of the streams is available. However, as this knowledge is usually very application-specific, in our experiments we decided to compare the schedules by using AWW. The rating function values AWW for the example in Figure 4.1 are shown in Table 4.1, where a lower value of the rating function indicates a better system performance.

4.3 Dynamic Offset Adaptation

In our approach, we propose the dynamic adaptation of offsets, which change over time following the change of the traffic on the CAN bus. Because the system is distributed with no single point of system status and operation knowledge, the decisions to change offsets are based on traffic monitoring carried out by the individual nodes, independently each from the other. The two main differences between our proposed approach and the staticOAA [GCGCF08] are that (1) we consider the entire distributed system, where offset adaptations are carried out on all nodes dynamically and independently, and (2) the offset adaptation on each node is initiated periodically and offset changed only if the conditions of the traffic require the change. In the staticOAA approach all offsets are calculated in advance and do not change during system lifetime.

The details of the algorithm as well as an example that illustrate its operation are presented in Section 4.3.1. In Section 4.3.2, we compare the use of DynOAA with previous approaches qualitatively.

4.3.1 Dynamic Offset Adaptation Algorithm (DynOAA)

The DynOAA is run on each node independently and periodically at certain instances of time that will be discussed in Section 4.3.1.2. An illustration of the operation of the DynOAA for one stream is shown in Figure 4.2. In the upper part of the figure, on the top of the time line, the periodically released messages of the stream are indicated by small arrows. The larger arrows on the

scenario	AWW
fully asynchronous	0.18
synchronous/asynchronous	0.084
synchronous, maximum	0.030
synchronous, average	0.003
optimal	0

Table 4.1: Rating function values for approaches from Figure 4.1

bottom of the time line indicate the instances when the adaptations start or when DynOAA is run, which includes a period of traffic monitoring. In Section 4.3.1.1, we will first look into how the adaptation (new offset calculation) is done and in Section 4.3.1.2, we will explain the adaptation triggering process.

4.3.1.1 Adaptation

Before the adaptation takes place, the bus is monitored by each stream for the time equal to the maximum period of all streams in order to have enough time to characterize the recent traffic on the bus. A list, from now on called *busy_idle_list*, is created. An example of it is shown in the lower part of Figure 4.2. It contains for each time slot during the monitoring an idle element if the bus is idle and a busy element if the bus is busy. If not mentioned otherwise, a time slot has the length of the transmission time of one CAN bit. From the busy_idle_list, we can find the *longest idle time (LIT)* and *longest busy time (LBT)*, which are the maximum continuous intervals when the bus was idle or busy, respectively. When finding these intervals, we consider the busy_idle_list as a circular list (by considering the first and the last time slot adjacent). The variable *last_message* denotes the amount of time passed between the current time and the time the last message was released for this particular stream. This value is needed to calculate when the next message would be released. The *next_position* is the time that indicates when in the next cycle a message should be scheduled. It is chosen in the middle of the LIT. The next message of the stream is then delayed, i.e., the offset is adjusted, so that a message is released at the time specified by next_position.

A pseudocode that describes the adaptation of the offsets is shown in Algorithm 4.1.

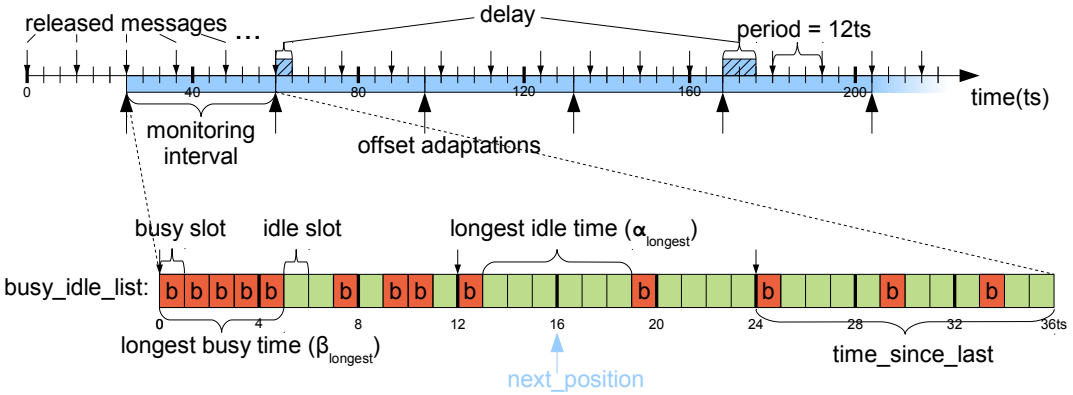


Figure 4.2: DynOAA illustration - timing diagram and busy_idle_list on a single node.

Algorithm 4.1 Offset adaptation

```
1: monitor_time = 0
2: while (monitor_time ≠ max_period - 1) do
3:    if (current_timeslot = busy) then
4:       busy_idle_list.add(busy)
5:    else
6:       busy_idle_list.add(idle)
7:    end if
8:    monitor_time = monitor_time + 1
9: end while
10: if (first slot of LBT is this stream (see Section 4.3.1.2)) then
11:    next_position = position in the middle of the longest_idle_time
12:    delay = (next_position + last_message) mod period
13: else
14:    delay = 0
15: end if
```

4.3.1.2 Time of Adaptation

In distributed systems, all streams are considered independent from each other. If more than one stream starts to execute the adaptation simultaneously, there is a high probability that the value of the next_position at more than one stream will be identical. Instead of spreading, the message release times would in that case be clustered around the same time instance. Therefore, we need to ensure only one stream is adapting its offsets at the same time. Ensuring that only one node is adapting is achieved if all nodes make the decision whether to adapt or not based on a unique criterion based on the same information, the traffic on the bus in this case. The criterion we use is to select the stream belonging to the first busy slot of the LBT. The idea is that this stream causes the biggest delay, because it potentially could have delayed all subsequent messages in the busy period and therefore should be moved first. If there are more than one LBT, of equal length, the first one is chosen. If the monitoring phases of all nodes are synchronized, this mechanism ensures that all nodes choose the same stream.

4.3.1.3 Example

The algorithm is best explained by the example from Figure 4.2. A stream with a period of 12 time slots (ts) is considered. The transmission of one message is assumed to take one ts. The algorithm passes through two phases. In the first phase, monitoring, the busy-idle list in the lower part of Figure 4.2 is created. The first message of the LBT belongs to the stream under consideration. There-

fore, in the second phase, it will adapt, i.e., the next message release by this stream will be delayed. In order to calculate the needed delay, the next position is determined first by choosing the middle of the longest idle time, which is in this case 16 ts. The delay is then calculated as

$$\begin{aligned} delay &= (next\_position + time\_since\_last) \mod period \\ &= (16\,ts + 12\,ts) \mod 12\,ts \\ &= 4\,ts \end{aligned} \tag{4.2}$$

This delay causes the releases of this stream in the next hyper period to be shifted by additional 4 ts. For the messages at positions 1 to 4 this means, if they were delayed, their response time is reduced. The probability that a message will be delayed depends on the number of preceding messages. If the message is not preceded by any other message, the message cannot be delayed. Therefore, it makes sense to choose the first message from the LBT to adapt.

4.3.2 Qualitative Comparison

In Table 4.2, different approaches for scheduling and analyzing CAN are compared against the following properties:

flexibility: It takes into account whether the approach can only cope with fixed scenarios, or it is flexible enough to handle addition or removal of streams during run-time.

response times: The length of the response times of messages are very important because if they are too long the system might fail.

computation/communication overhead: This describes how often and how much additional computation/communication the approach requires.

predictability: It shows whether it is possible to calculate bounds for the actual response times.

The schedulabilty analysis and random offset assignment approaches use the basic CAN access mechanism allowing random access of the streams. By using schedulability analysis, however, pessimistic worst case response times are calculated and used as upper bounds. The StaticOAA approach goes a step further and assigns static offsets between the streams on one ECU. The clock on the ECU guarantees that these offsets remain fixed and analysis can be used to calculate worst case response times. The DynOAA approach extends the assignment of offsets to the whole distributed system, adapting the offsets dynamically. A different strategy is to set a protocol on top of CAN. FTT-CAN

Table 4.2: Qualitative comparison of different CAN scheduling strategies

	schedul. analysis [DBBL07]	random offset assignment	staticOAA [GCGCF08]	DynOAA	FTT-CAN [APF02]
flexibility	no	yes	no	yes	yes
response times	bad	good	average	very good	very good
communi-cation overhead	no	no	no	no	yes
computation overhead	offline low	no	offline low	online low	online high
predict-ability	yes	no	yes	partial	yes

[APF02] is in our opinion the best option, because it provides the flexibility of scheduling both event-triggered and time-triggered message streams.

For the first three properties, DynOAA is better or equal to the previous works. DynOAA has full flexibility similar to random offset assignment. FTT-CAN is also able to add and remove streams at run-time, but it incurs communication overhead due to registration at the master node. The message response times of DynOAA are shorter compared to the first three strategies using the AWW as showed in the following section. Because of the mixed time- and event-triggered scheduling of FTT-CAN, it is difficult to compare the response times with DynOAA. The event-triggered streams have response times similar to the random offset assignment, while the time-triggered streams have zero response times if the tasks are executed according to the FTT-CAN schedule. The communication overhead of DynOAA cannot be further reduced as there is none. The two properties we have to take a closer look at are: computation overhead and predictability. The schedulability analysis and staticOAA offsets are determined before system deployment and actually do not introduce any run-time overhead. DynOAA and FTT-CAN are based on run-time calculations. Using FTT-CAN, a single master node calculates the schedule for the whole network, while DynOAA is executed on all nodes calculating only the local behavior that reduces the amount of computation needed drastically. The predictability when using FTT-CAN is better, because it supports time-triggered access. Schedulability analysis and staticOAA are based on analytical models and, therefore, can provide guaranteed WCRTs while our approach lacks analytical analysis and guarantee. However, our approach is not as unpredictable as a purely random approach, as we demonstrate by experiments that show the schedule created by

DynOAA always improves over time and the actual response times will never reach the values calculated by the analytical approaches.

4.4 Experimental Evaluation

In the following, we will evaluate DynOAA by means of simulation using typical automotive scenarios. We initially use simulation, because it has the advantage that different parameters can be analyzed very fast. Additionally, simulation allows us to have a full view of the system at any time, so we can easily extract the performance values and trace situations of interest. The simulation described in Section 3.5 is used and extended. Compared to the simulation of PLA, following aspects have to be considered:

- Instead of specifying the scenario by hand, we implemented a file import that can read xml-files according to the specification of Netcarbench [BHN07].
- A consequence of the discrete event simulation is that only the synchronous case, where all nodes have the same time base, can be simulated. This means, if the offsets are fixed and not adapted by DynOAA, the schedule repeats after time equal to the hyper period. The asynchronous case is then modeled and simulated by using different randomly selected initial offsets.
- The adaptation value returned by the Adaptation object depicted in Figure 3.24 is the time the stream should be delayed.
- Even though one simulation step is one CAN bit, we can simulate the asynchronous behavior due to clock drift by adjusting the periods of the streams in a much smaller step size. Concretely, we use 32 bit floating point numbers to represent the period. This has the effect that is similar to a real setup, the shift of the message by the clock drift is reflected when it passes over the length of one CAN bit. However, first we take a look at the synchronous case.

4.4.1 Scenarios

For our experiments, we used scenarios common in the automotive domain as described in Table 4.3. The scenarios are split into three groups: (1) Two scenarios provided with the RTaW-Sim [RS09] (ls_50, hs_50a), (2) an example provided by Herpel et al. [HHKG09] (hs_50b), and (3) four synthetic scenarios generated by Netcarbench [BHN07](hs_60 to hs_90). Netcarbench allows generating typical automotive scenarios for a given workload. We generated scenarios with a

bus speed of 500 kbit/s with different average loads. The nomenclature for the scenarios is the following. Scenarios with a CAN bus speed of 125 kbit/s start with *ls* (low-speed) and with 500 kbit/s start with *hs* (high-speed). The number at the end describes the average bus load of the scenario. For example, hs_60 is a scenario with the bus speed of 500 kbit/s with a workload of 60 percent. All scenarios are compared in terms of AWW defined in Section 4.2.

4.4.2 Adaptation Memory

A disadvantage of the proposed adaptation criterion is that the adaptation can oscillate between the same positions in certain situations, when the schedule cannot be improved further because the same offsets are repeatedly tried. An example is shown in Figure 4.3, where stream 1 is scheduled back and forth between schedules 1 and 2. These oscillations can also happen over several adaptation steps. However, it can easily be avoided if each stream records its last q chosen delays in the adaptation memory, effectively meaning that each stream has to use q memory locations to store these delays. In each subsequent adaptation step, instead of taking one of the previously tried delays, the delay is decreased by one message length. The resulting behavior will be that the stream, in most cases, is not at the beginning of a longest busy period, and another stream will be chosen for adaptation. In schedule 3 in Figure 4.3, the alternative next_position is shown.

We did several experiments with different sizes of adaptation memory. For example, Figure 4.4 shows the rating of the current hyper period over time using DynOAA with different q starting with random offsets. If no oscillation occurs, the schedule improves to a rating of zero where all message response times are zero. Otherwise, the rating value stagnates when the oscillation happens. From the experiment, we can conclude that when larger memory is used, the rating of the schedule decreases further and therefore, the probability of oscillations

Table 4.3: Scenarios used for the experiments

scenario	speed (kbit/s)	no of streams	workload	max period	source
ls_50	125	145	0.51	2 sec	[RS09]
hs_50a	500	85	0.51	2 sec	[RS09]
hs_50b	500	56	0.48	1 sec	[HHKG09]
hs_60	500	97	0.60	2 sec	[BHN07]
hs_70	500	101	0.71	2 sec	[BHN07]
hs_80	500	109	0.81	2 sec	[BHN07]
hs_90	500	131	0.90	2 sec	[BHN07]

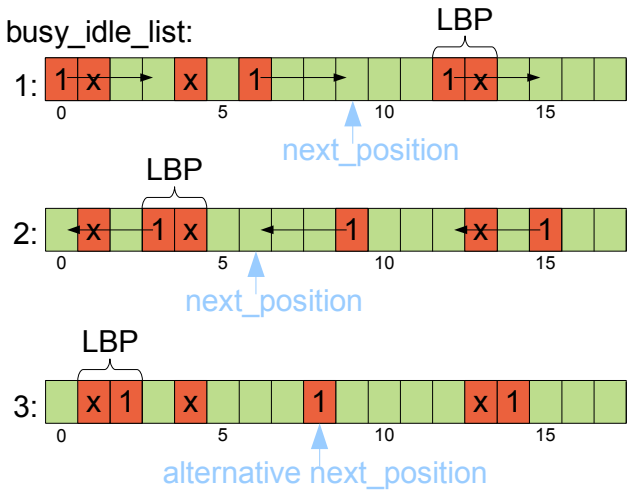


Figure 4.3: Example where the adaptation gets stuck because it is repeatedly trying the same schedules. 1: Initial busy_idle_list, 2: regular adaptation, 3: alternative adaptation from initial state.

decreases. However, the relationship between the increment of memory increase and probability of oscillations is not linear. Further investigation of this phenomenon is left for future work, because the described problem only occurs for synchronous systems with periodic streams. Therefore, for all the following experiments we used an adaptation memory of the maximum integer value that the simulation engine supported ($2^{31} - 1$).

How often the algorithm reaches an oscillating state and the time until leaving that state is difficult to predict. We have found in the experiments that using a small memory with 10 entries would already be sufficient to avoid oscillations for our test scenarios.

4.4.3 Rating over Time

Figure 4.5 shows the AWW that uses the maximal response times recorded since the start of the simulation. The experiments were always run with 10 different random offset initializations. The continuous lines in the plots represent the average of these 10 runs, while the vertical error bars indicate the maximum and minimum value of the rating function at that instance of time. These values are comparable to the WCRTs of the analytical analysis [GHN08, DBBL07], as they reflect the worst response time that is found during our simulation. The AWW for the different approaches are listed in Table 4.4. Even though we cannot prove that the WCRTs have reached their maximal value, the simulation

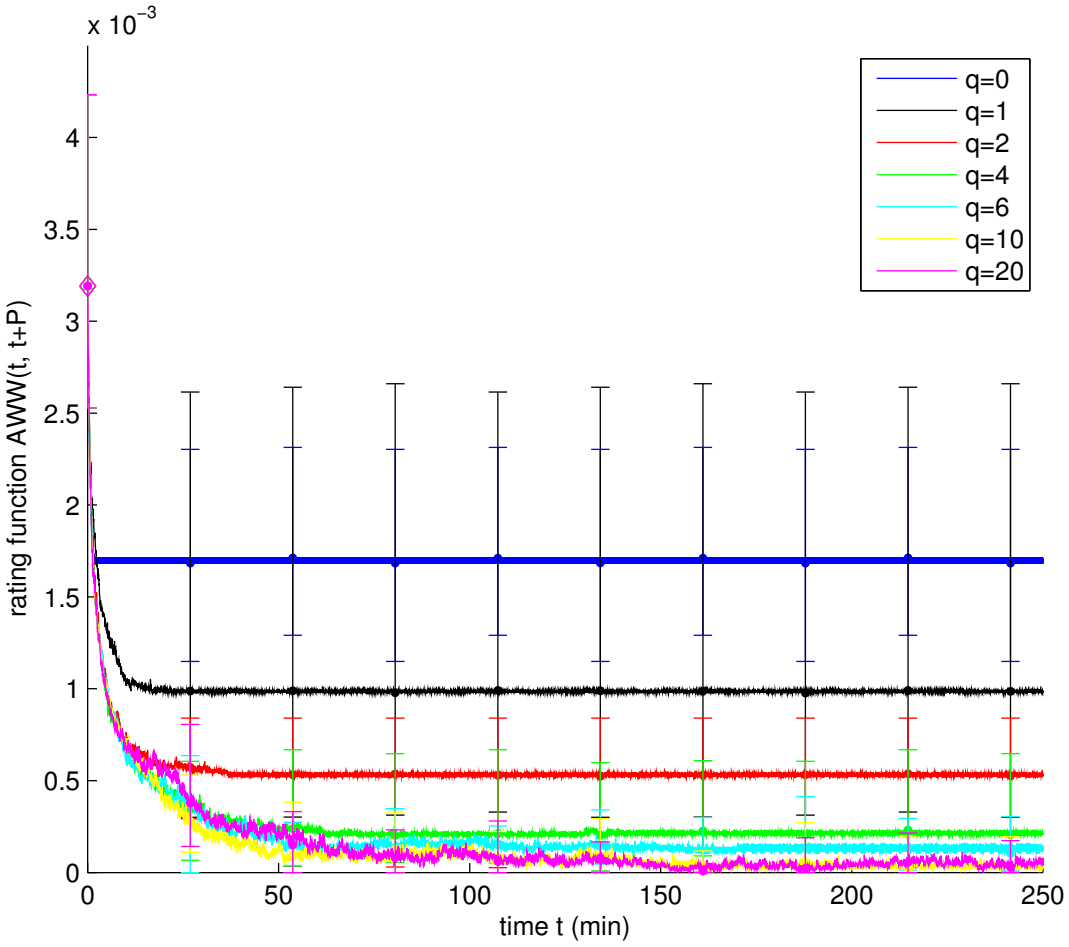


Figure 4.4: Average of AWW of 10 runs with different initial random offsets as a function of time for different sizes of memory for scenario ls_50. The error bars denote the minimum and maximum values. The legends are always in the same top-down order as the graphs.

experiments show that the rating function is increasing very slowly over time for a reasonable amount of simulated time (1400 min≈1 day).

Figure 4.6 shows the rating function only for the last adaptation interval (2 seconds in this case). We can see that it converges very fast to a stable value. The plot also shows that we are always improving compared to no adaptation case which is represented by the rating values at time zero (diamonds). If we are dealing with a soft real-time system, where a few long response times can be tolerated, our method offers outstanding performance.

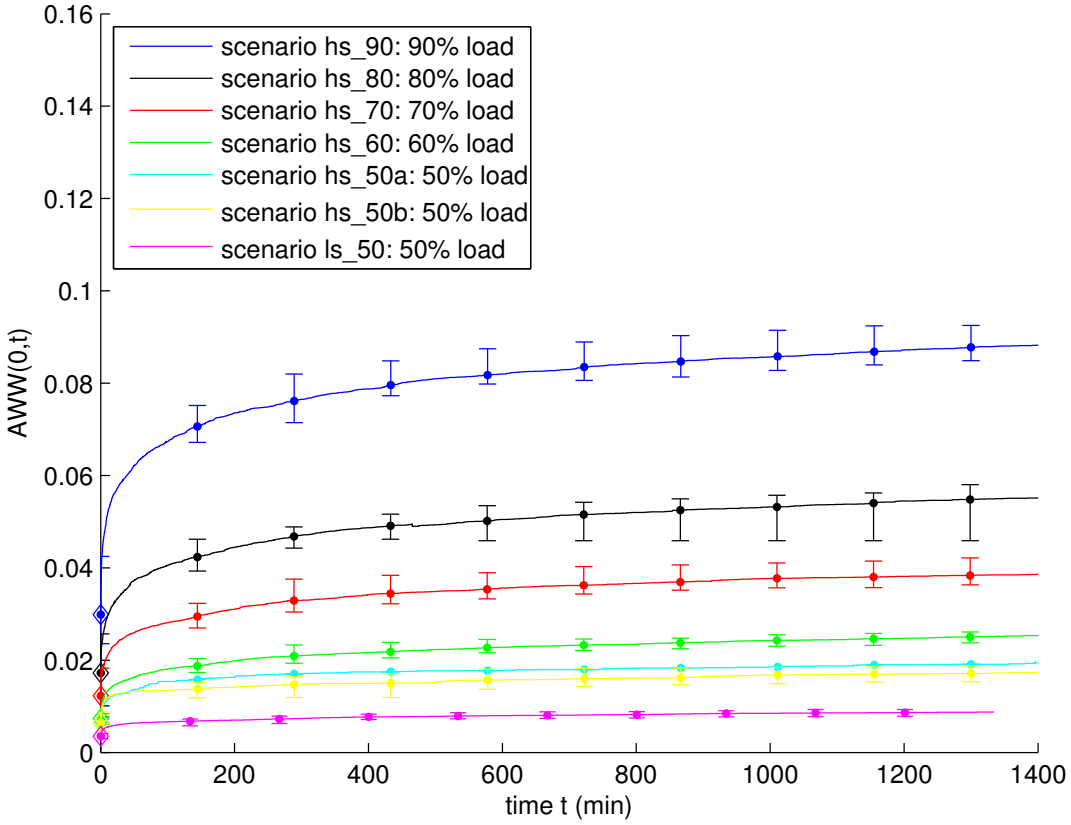


Figure 4.5: Rating function AWW as the function of time - maximal response times for the entire simulation taken into account

scenario	Analytical [DBBL07] ($\cdot 10^{-1}$)	StaticOAA [GHN08] ($\cdot 10^{-1}$)	DynOAA (entire sim) ($\cdot 10^{-1}$)	DynOAA (average) ($\cdot 10^{-3}$)
ls_50	1.9	0.79	0.08	0
hs_50a	1.6	0.72	0.20	0
hs_50b	1.3	n/a	0.19	0
hs_60	1.4	0.63	0.26	1.9
hs_70	1.7	1.03	0.43	4.0
hs_80	2.4	1.10	0.59	6.6
hs_90	3.0	2.84	0.95	12

Table 4.4: AWW for different scenarios, where the first and second column take into account the WCRTs of the analytical worst case, the third column of the entire simulation time and the last column shows the average AWW for the last hyper period.

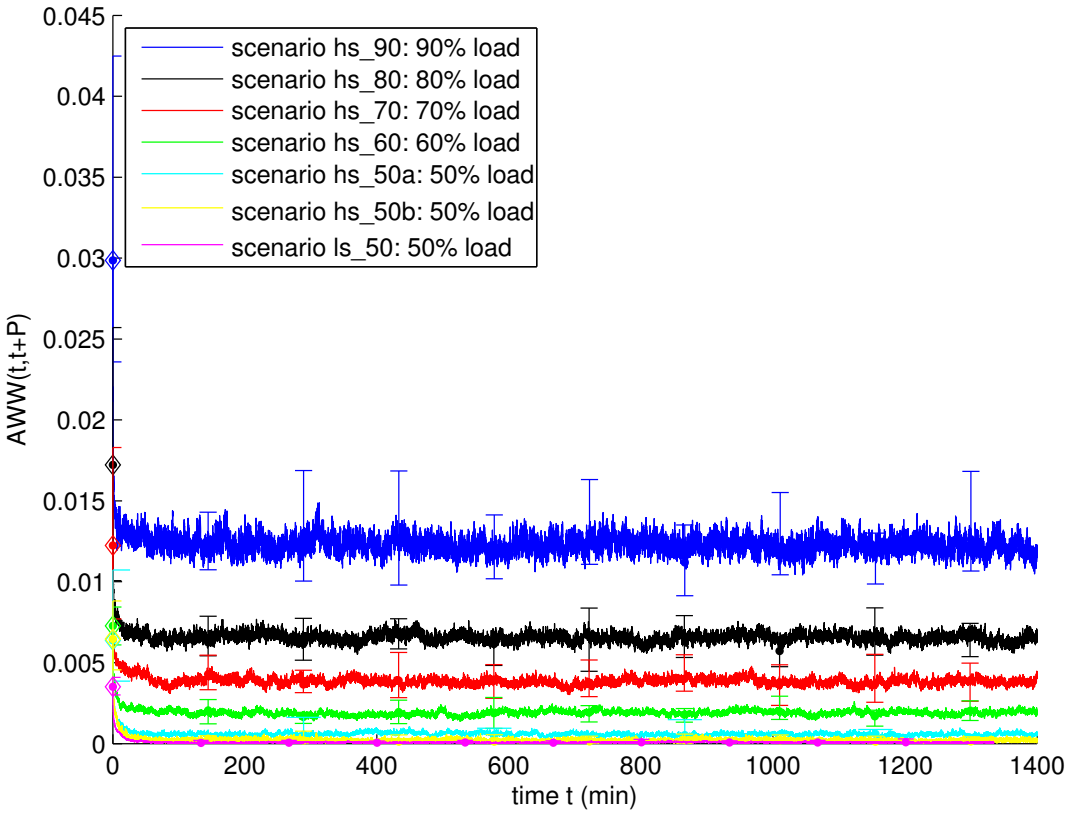


Figure 4.6: Rating function AWW as the function of time for different application scenarios - response times of the last adaptation interval taken into account

4.4.4 Asynchronous Monitoring

In the previous sections, we assumed the monitoring intervals were synchronized. However, in a distributed system, the monitoring intervals of the streams are asynchronous because of the clock drift [MNB11]. This makes choosing the adapting stream more difficult. We cannot guarantee anymore that exactly one stream is adapting, as is shown in the example in Figure 4.7. In the example, the monitoring phase of the two streams overlap in such a way that the longest idle time is the same, but the longest busy period is different. Therefore, streams 1 and 2 would schedule their messages to the same position. This only occurs due to the temporary different monitoring intervals during adaptation. Hence, given the monitoring intervals have the same length, the maximal number of streams scheduled to the same idle time is two. This means the effect is not severe as the experiments in the following will show.

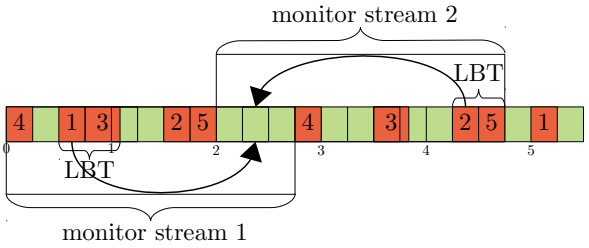


Figure 4.7: Example where asynchronous adaptation leads to scheduling of two stream to the same position.

Another problem with asynchronous monitoring occurs when there are several longest busy periods with the same length. If the streams always choose the first LBT, it can lead to a stop of adaptation, because no stream will adapt. How this can happen is shown in the example in Figure 4.8. On the one hand, this problem can be simply avoided by choosing randomly the LBT that is considered. On the other hand, if the monitoring periods are synchronous, the random selection can cause that several streams adapt simultaneously to the same position.

In Figure 4.9, the rating over time for scenario ls_50 is depicted. The meaning of the error bars is similar to the one described for Figure 4.6. Table 4.5 shows the average rating values for the cases from Figure 4.9. The results for asynchronous monitoring, and choosing which stream to adapt based on the first LBT are not displayed, because, as described above, the adaptation in the simulation gets stuck. As expected, the setup with synchronous monitoring and choosing which stream to adapt based on the first LBT performs best. However, the difference in performance is negligible considering the variation within the results. Results with other scenarios, not shown here, reveal a similar behavior, leading to the conclusion that DynOAA also works well for asynchronous monitoring and the afore mentioned effects can be neglected.

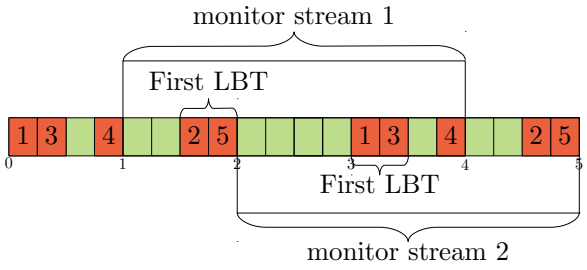


Figure 4.8: Example where the adaptation stops because no stream is first of the first longest busy period.

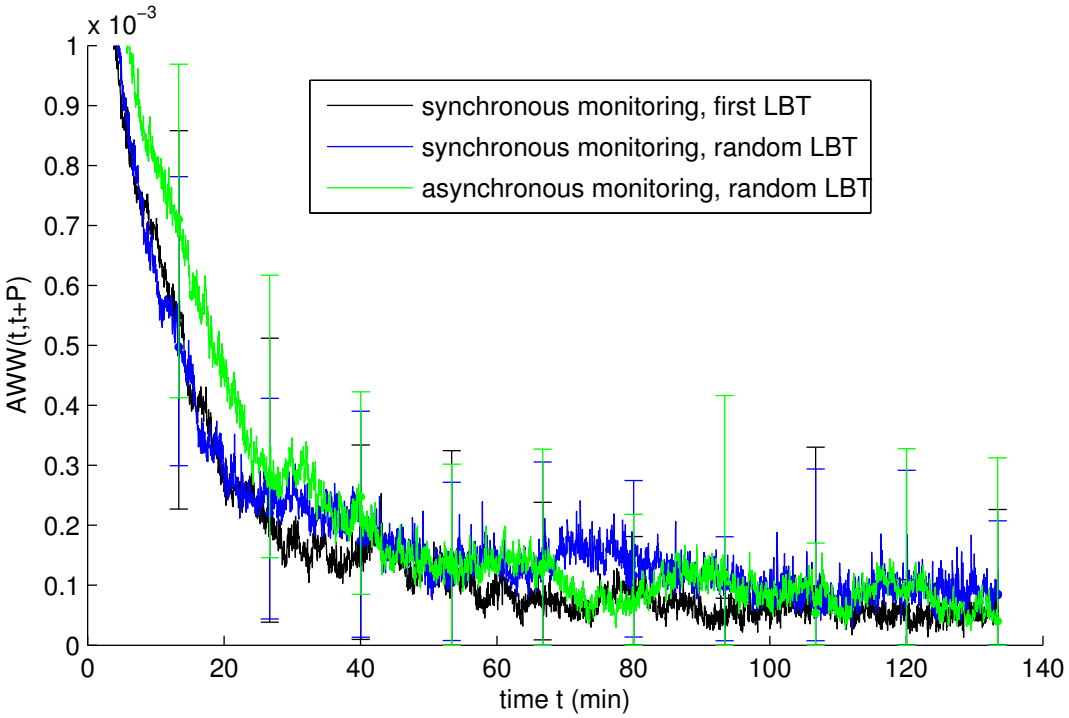


Figure 4.9: Rating function as a function of time, comparing synchronous to asynchronous monitoring and choosing the first LBT to choosing a random LBT for deciding which stream adapts.

synchronous monitoring, first LBT	$2.0011 \cdot 10^{-4}$
synchronous monitoring, random LBT	$2.3788 \cdot 10^{-4}$
asynchronous monitoring, random LBT	$2.5792 \cdot 10^{-4}$

Table 4.5: Average rating value over the whole simulation time for the scenario in Figure 4.9.

4.4.5 Adaptation in Dynamically Changing Systems

In the following, we model the dynamic changes of the system configuration by allowing addition and removal of streams during system operation. We start with the system operation with a fixed number of streams. When a stream is removed from the system, the response times of the remaining streams will either decrease or stay the same, depending on whether the removed stream delayed the transmission of another stream or not. Similarly, when a stream is added, the response times of the other streams increases or stays the same depending on whether the added stream delays another stream or not. The goal of the adaptation, keeping the response times as short as possible, is achieved

by adapting the offsets dynamically and finding a new set of offsets for the current scenario that does not depend on the set of offsets for the case before the change of system configuration. We demonstrate this behavior by running DynOAA for different initial offset assignments and comparing the rating value after the system has converged. The results of experiments in Figure 4.5 (see Section 4.6) show exactly this, because the system converges to a certain rating value depending on the used scenario and not on the initial offset assignment.

Because the adaptation is constantly performed, the rating value fluctuates even when the system is not changing configuration. Therefore, the reaction on adding or removing one stream will not be obvious. However, we can show the system reaction when several streams are added or removed. In Figure 4.10, we show the scenario in which 64 streams are abruptly added to the system at time 100 min, and then removed at time 200 min. Corresponding system load increased from 50% to 70% and back to 50%, respectively. Figure 4.11 illustrates the system behavior when the change was not abrupt, but rather gradual. From time 100 min, 64 streams are added to the system at the rate of one stream per minute, and then after time 200 min the streams are removed from the system, one stream per minute. This shows that the DynOAA adapts offsets and stabilizes the value of the rating function in each case for each new configuration.

4.5 Extension to Multi-segment Buses

Data rate limitations of CAN can be overcome in some applications by using multi-segment systems. In the following, we present the case of the system where the segments are connected through a single gateway. Firstly, multiple segments can help to increase the capacity of priority-based communicating embedded systems. Secondly, even though DynOAA reduces the message response times to a minimum for a given load, the multi-segment approach can potentially reduce the load of the individual segments and thus reduce the message response times further. We consider a simple case where a single segment is split into up to five segments as this represents the current scenarios in the automotive domain. Gains can be expected if the inter-segment communication is not too intensive. Finally, the reason for a multi-segment approach can be purely practical. For example, in the development of large distributed systems, which consist of multiple application domains, the systems are designed by multiple teams whose efforts cluster around individual domains each implemented on a single-segment system. The integration of the overall system then becomes easier as it only requires connection of multiple segments via a gateway.

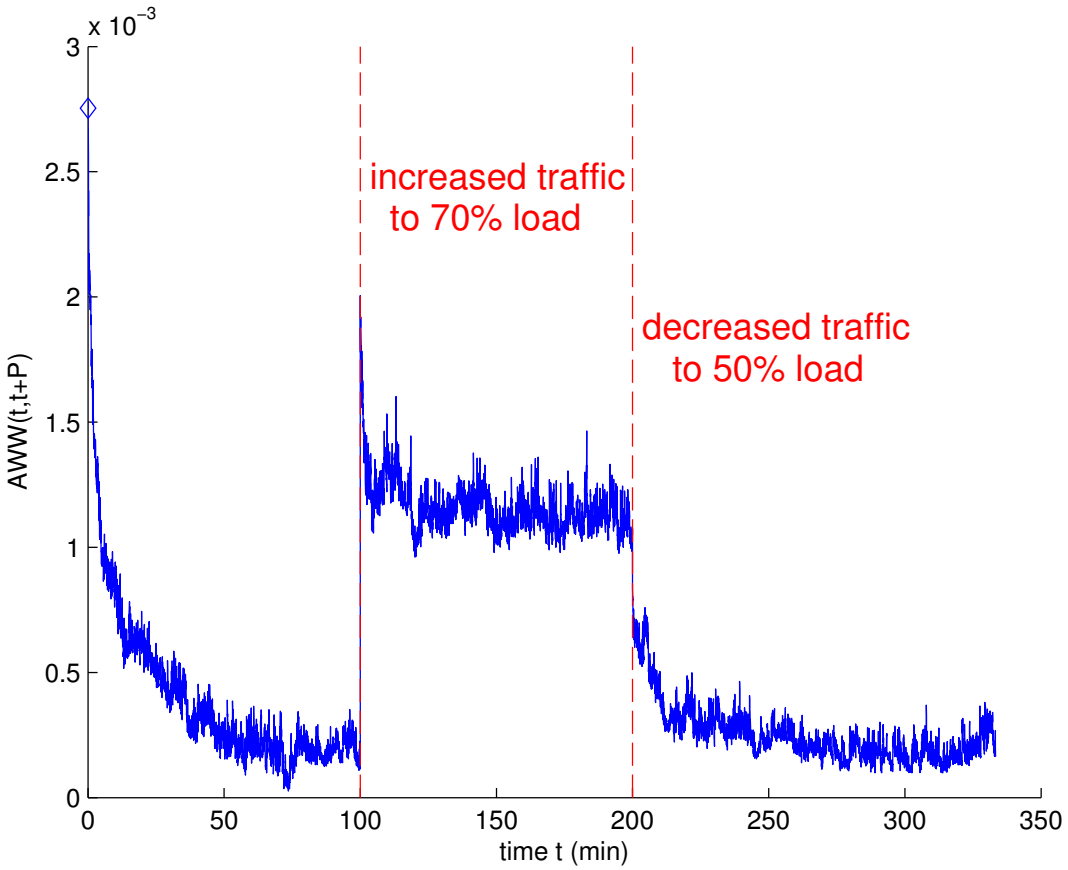


Figure 4.10: Rating function as a function of time for a scenario in which at time 100 min, 64 streams are added such that the load increased from 50% to 70%. At time 200 min, these streams are removed again.

4.5.1 System Model

In the following, the system model described in Section 2.1 is extend by allowing d buses connected by a gateway. Each stream is connected to exactly one bus and can send messages to one or multiple buses, as depicted in Figure 4.12. Therefore, a multi-segment stream s'_i is characterized by a tuple $(T_i, O_i, \sigma_i, \omega_i)$, which contains additionally to a single-segment stream (cf. Section 4.1) a source bus σ_i, to which it is connected and a set of destination buses ω_i. The response times of a multi-segment stream are the times between a message release and the start of its uninterrupted transfer on the destination buses. The worst case response time is, analog to the single-segment case, the largest of these response times. A message that is transmitted on a different bus than the source bus will be called a *routed message*. For the gateway, we make the following assumptions:

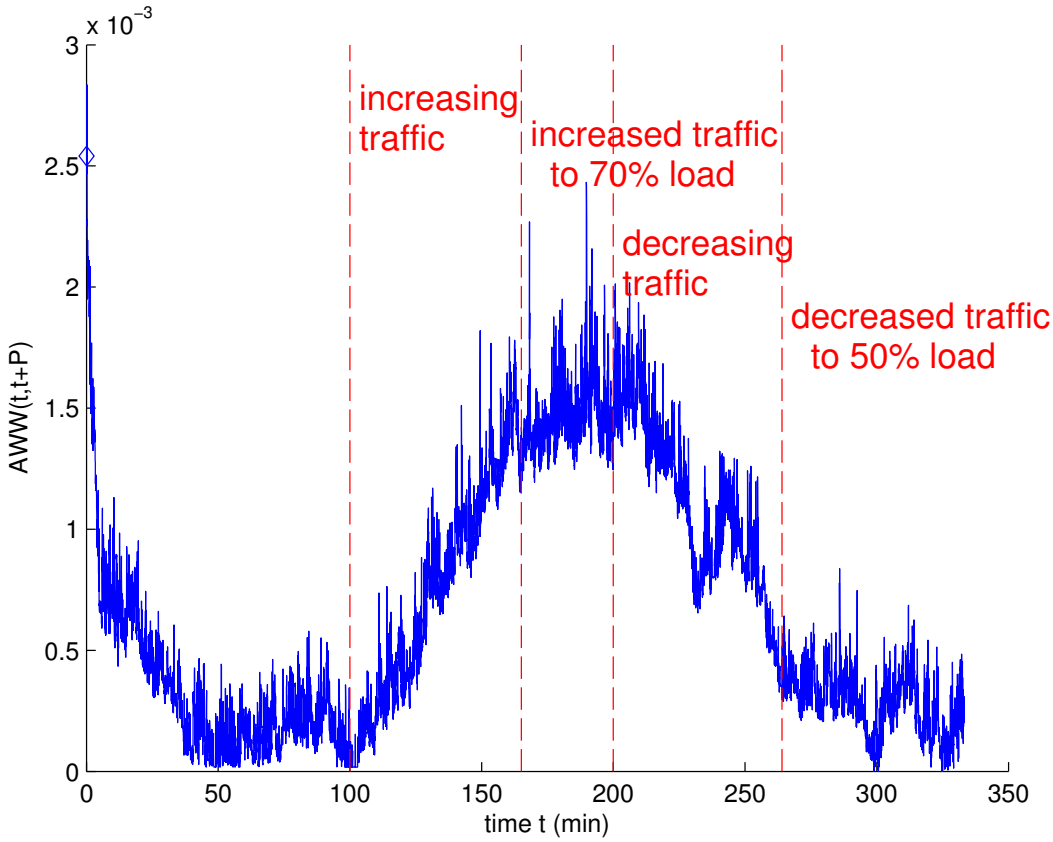


Figure 4.11: Rating function as a function of time for a scenario in which at time 100 min, 64 streams are added one every minute such that at time 164 min, the load is increased from 50% to 70%. At time 200 min, these streams are removed again one every minute.

- The gateway is central, so it is connected to every bus. It has to be ensured that all connection requirements of the streams are fulfilled. Delays caused by computational load on the gateway are neglected. This is a reasonable assumption, because the operation speed of the gateway is multiples of the communication protocol.

- The gateway uses priority-based transmission with unlimited buffers. This means, in contrast to a first-in-first-out strategy, when several routed messages are pending for transmission, the message with the highest priority will get transmitted.

- A routed message will be pending as soon as the complete message on the source bus has been transmitted.

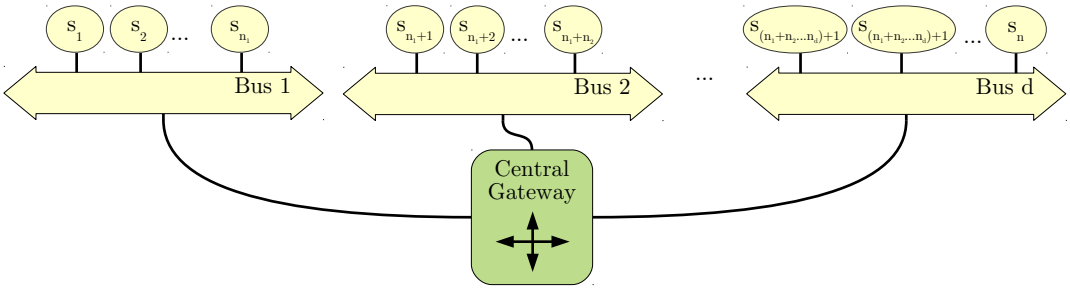


Figure 4.12: System model for a multi-segment Controller Area Network

4.5.2 Partial Adaptation

An obstacle for applying DynOAA to multi-segment systems is that the current traffic information for the whole system is no longer available to each node, because a certain fraction of messages is routed by the gateway. This means that the nodes change their offsets based only on information on local traffic of the segment they are connected to. Section 4.5.3 will demonstrate the influence of this on the system performance.

Another problem in the multi-segment case is that it is not possible to influence the release of all messages on a segment directly. For example, messages received from the streams of the other bus segments are not aware of the traffic situation on the receiving bus segment and cannot perform correct adaptation. The knowledge of the full traffic situation would be possible by using the gateway to collect the traffic information and transmit (broadcast) it to all nodes on all segments, but it would result in additional overhead and require additional bandwidth. Therefore, this option is left out in our work.

Based on the above analysis, our decision was to first analyze the case in which only a part of the streams, belonging to the messages on one segment, perform DynOAA and adapt their offsets, while the others are not doing that, and we call this partial adaptation. This type of behavior can then be extrapolated to the multi-segment systems directly. The problem with the partial adaptation is that the DynOAA can select a stream that does not adapt as one that should adapt, and the intended goals would not be achieved. In the case of our synchronous simulation, it even can lead to complete stop of the adaptation. However, a slight modification of the DynOAA corrects the algorithm operation and makes it suitable for the partial adaptation case. The solution is outlined in the following paragraphs.

During traffic monitoring, in addition to the information whether a slot is busy or idle, we record the information whether the busy slot is allocated by an adapting or non-adapting stream in the *annotated busy_ idle_ list*. This information can be provided by using an indicator (flag) associated with the current

message with two possible values, adapting or non-adapting. This indicator could be statically assigned to each message stream at design time, but the flexibility of dynamic adaptation and self-organization would be lost in that case. Therefore, we propose to use the last (least significant) bit of the CAN identifier to assign a priority to the stream to indicate whether it is adapting or not. This has the additional advantage that there is no need to keep the list of all streams and at the same time allows flexible insertion and removal of nodes from the system. The penalty for the adopted approach is that the number of available identifiers is halved. In our opinion, this is not a big disadvantage because the number of still available identifiers is large enough to cover the needs of real distributed systems. The number of available identifiers becomes $2032/2 = 1016$, which is large enough to represent real-life systems. For example, in the VW Golf VI (2010 model), 49 nodes with an average of 140 streams are used [Rac10]. In addition, due to the limited capacity of the CAN bus, the number of needed identifiers will most probably not grow.

From the annotated busy_idle_list, each node again calculates the LIT and LBT. The longest idle time is determined similar to the description given in Section 4.3.1. As for the LBT, we choose the maximum continuous interval in which the bus was busy starting with an adapting busy slot. This way we ensure that the first slot of the LBT belongs to an adapting stream and, therefore, the adaptation is done. This stream also represents the adapting stream that potentially delays most other streams. In the example in Figure 4.13, even though the second busy time has four adapting busy slots, the first slot of the first busy time, i.e., here LBT, potentially delays four time slots, while the first one of the second busy time only delays three slots. Therefore, it is better to adapt the first, as it is chosen by the algorithm.

In Figure 4.14 and 4.15, it is shown how the enhanced DynOAA performs for different numbers of adapting streams (and nodes) expressed as a fraction of total number of streams. The experiments carried out for this case are similar to the experiments from Figure 4.6. For the scenario ls_50, if none of the streams are adapting, the rating value in average is constant at 0.00376. If 50% of the

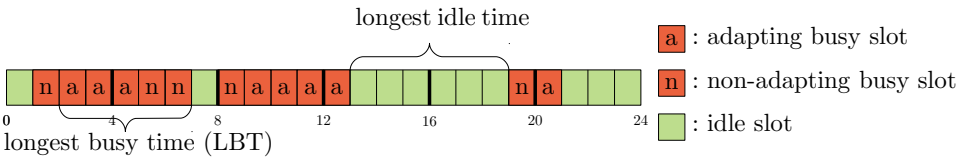


Figure 4.13: Example of an annotated busy_idle_list, storing for each time slot during monitoring whether it is idle or busy by an adapting or non-adapting stream.

streams are adapting, the rating value converges to an average of 0.00124. This is less than a half of the value of the non-adapting ones:

$$0.00124 < \frac{0.00376}{2} = 0.00188.$$

For the scenario hs_70, this observation is strengthened further. The performance difference between all (100%) adapting and 75% adapting streams is very small. In addition, the experiments show that the time needed to converge to a stable state depends on the number of streams that are adapting. The converged state is reached faster if fewer streams are adapting.

4.5.3 Experimental Results

The conditions to run DynOAA on multi-segment systems, by allowing some messages not to adapt, have been outlined and established in Section 4.5.2.

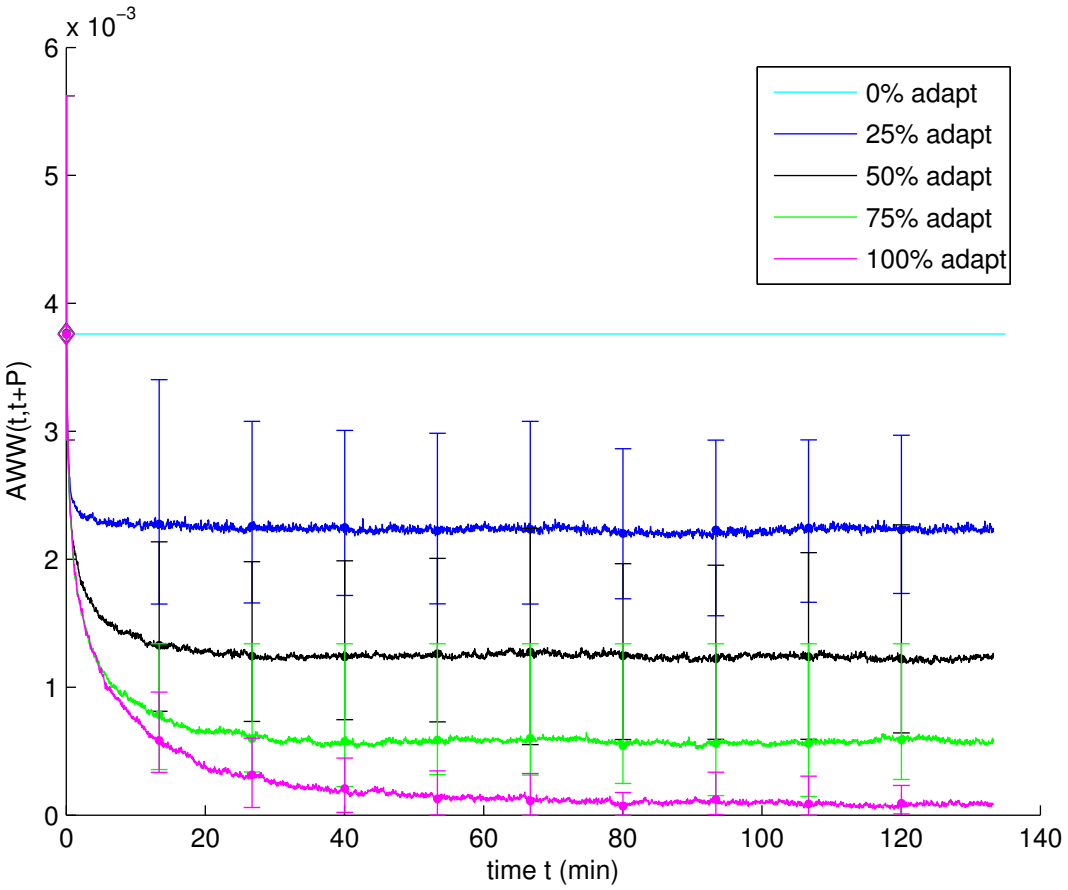


Figure 4.14: Rating function as a function of time for different fractions of streams adapting for scenario ls_50.

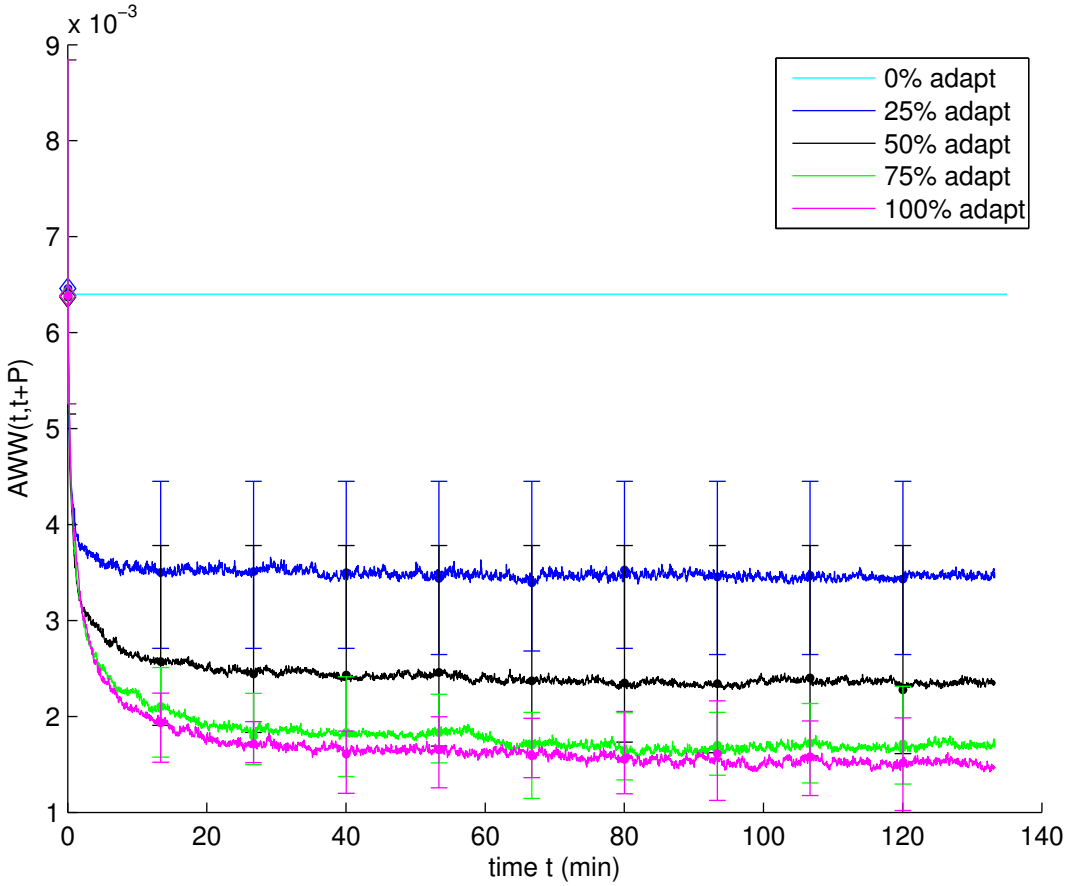


Figure 4.15: Rating function as a function of time for different fractions of streams adapting for scenario hs_70.

In the multi-segment system case, the routed messages are simply considered as non-adapting messages. In order to evaluate the performance of DynOAA for multi-segment networks, we generated new scenarios extending the single-segment scenarios by Netcarbench as described in Section 4.4.1 by providing additional information on the source and destination segments for each message stream.

We performed a number of experiments to analyze the performance of multi-segment systems using our rating function in Eq. (4.1). These experiments were performed under the conditions and assumptions outlined below. We first assumed that the source segment of each stream is assigned such that each segment contains an equal number of streams. The destination segment is characterized by two parameters, *percentage of routed segments* ψ_{routed} and *percentage of received segments* $\psi_{received}$. The first parameter ψ_{routed} specifies the percentage of streams that are routed at all, i.e., whether the destination bus is different

than the source bus. The second parameter $\psi_{received}$ specifies the percentage of segments, from all available segments, which are included in the destination segment set. Which stream is routed and which segment is chosen as the destination is determined randomly, excluding the case when the source and destination segment are the same. For example, for the case of eight streams in the system with four segments with $\psi_{routed} = 50\%$ and $\psi_{received} = 25\%$, two streams are assigned to each segment. One of these two streams has one other segment as its destination.

For the experiments presented in this section, the streams of the scenario hs_80 from Section 4.4.1 are used. The received parameter $\psi_{received}$ is set to 50%. In Figure 4.16 and 4.17, it is shown how the enhanced DynOAA performs for different multi-segment scenarios. The experiments carried out are similar to the experiments presented in Figure 4.6. These experiments show that when using DynOAA for the multi-segment systems, the rating value decreases, i.e., performance improves compared to random chosen values, as denoted by the values at time zero. The increase of the AWW with increasing percentage of routed signals ψ_{routed} is explained by the increasing workload per segment.

The experiments shown in Figure 4.18 are done under assumption of the same workload on each segment, because all streams from the initial set at every segment are either directly scheduled or are routed streams. First, it has to be noted that the AWW is increasing with the number of segments for DynOAA at the converged state, as well as for randomly set offsets at time zero. This is because the AWW for multi segments considers the worst response time of each stream on any segment and with increasing number of segments the likelihood of long response times increases. Nevertheless, the experiments show that using DynOAA reduces the AWW compared to random chosen values, as denoted by the values at time zero.

4.6 System Initialization

A typical CAN bus system begins its operation by turning on all nodes simultaneously. This procedure is most often very simple to implement, because one shared source of power is used. From scheduling perspective, this synchronous startup would be desirable, as we could plan the initial offsets a priori.

However, in practical implementations the boot and initialization times of the different ECUs are not equal. They even vary from one start to another, because initialization programs take different time depending on different factors including the conditions of the physical environment. For example, if it is very cold, the rain sensor needs longer to provide initial measurements compared to when it is warm. Therefore, we have to assume random initial offsets. However,

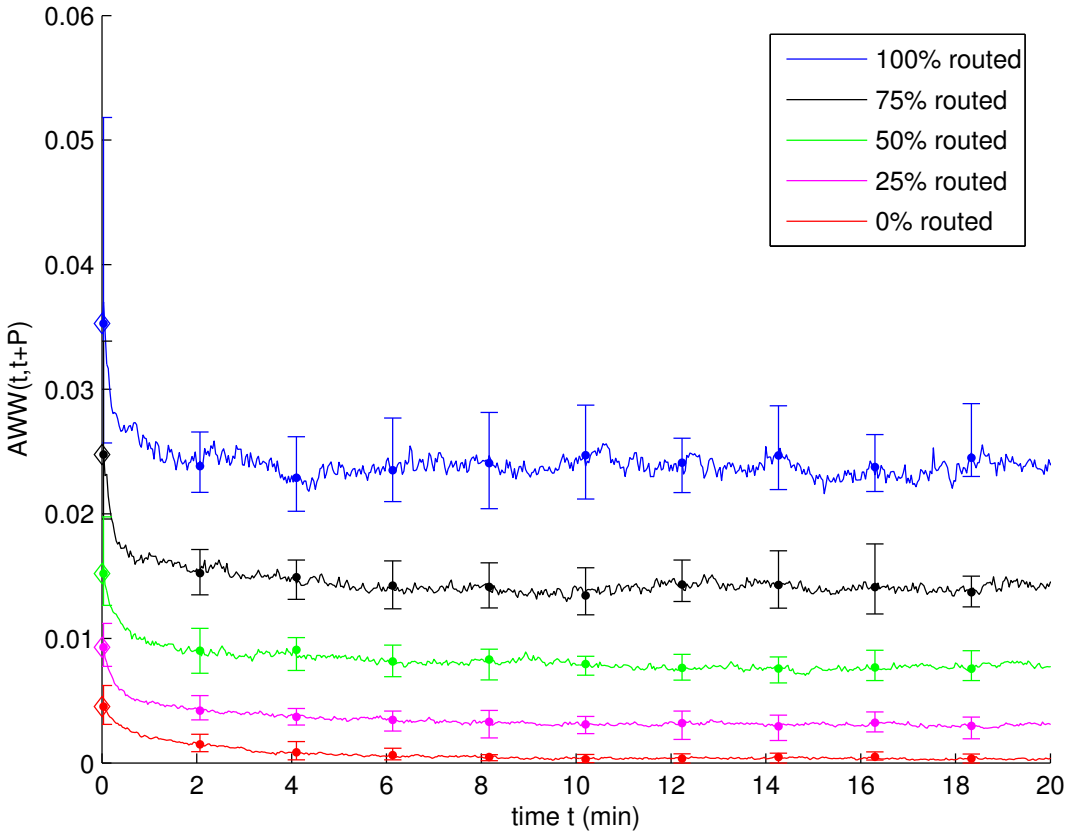


Figure 4.16: Rating function as a function of time for a scenario with two segments with different fractions of streams routed to the other segments (ψ_{routed}). $\psi_{received} = 50\%$.

if the initial offsets are set randomly, then the worst case for an offset free system becomes possible when all streams transmit at the same time.

One possible way to alleviate this would be to measure the startup times of each ECU under different environmental conditions. Using these measurements, we could assign offsets a priory. However, in this case it is not feasible to apply the methods for single processor scheduling [Goo03], because the offsets will not be fixed but represented with a range of values. Hence, a new method would be needed. In addition, the measurements will be very time consuming and could only be done very late in the system development phase. Also, any modification and revision in system design would require all offsets to be recalculated and changed on all ECUs. Therefore, we do not adopt this approach, but propose an algorithm that initializes the streams automatically not requiring any a priori knowledge.

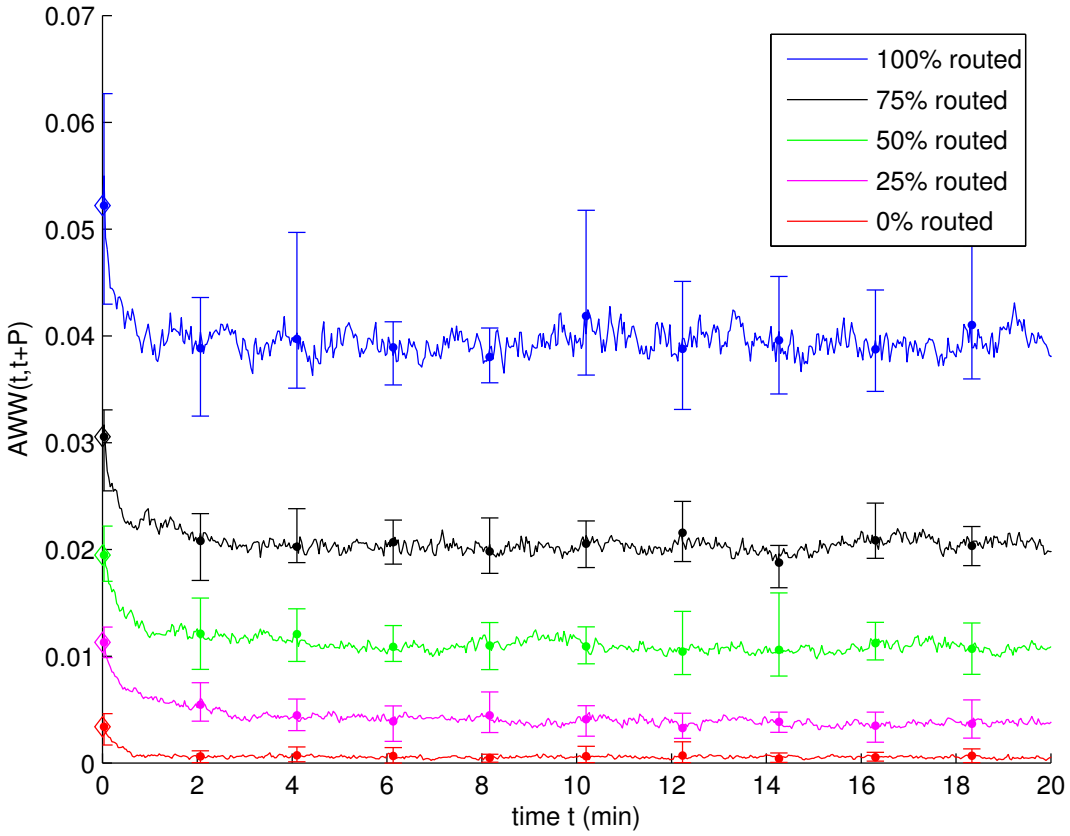


Figure 4.17: Rating function as a function of time for a scenario with four segments with different fractions of streams routed to the other segments (ψ_{routed}). $\psi_{received} = 50\%$.

4.6.1 Dynamic Initialization

The goal of the algorithm is to avoid the worst case in which all streams release messages at the same time, and to improve over a random initialization. The basic idea is that the streams start sequentially, one after another, in a predefined fixed order, in our case defined by stream priorities. They can start in either decreasing (highest priority first) or increasing (lowest priority first) priorities. This can be implemented in a very lightweight way, where each stream waits with the transmission of its first message until the stream with next higher (or lower) priority successfully transmits a first message. The problem can be that not all priorities are assigned to the streams, effectively meaning that a stream may be indefinitely waiting on transmission of a message with immediately higher (or lower) priority. To avoid this situation, all streams keep track of the priority of the last released stream. If the next expected priority is not assigned to any node, the stream with the next lower (higher) priority will start. The unused

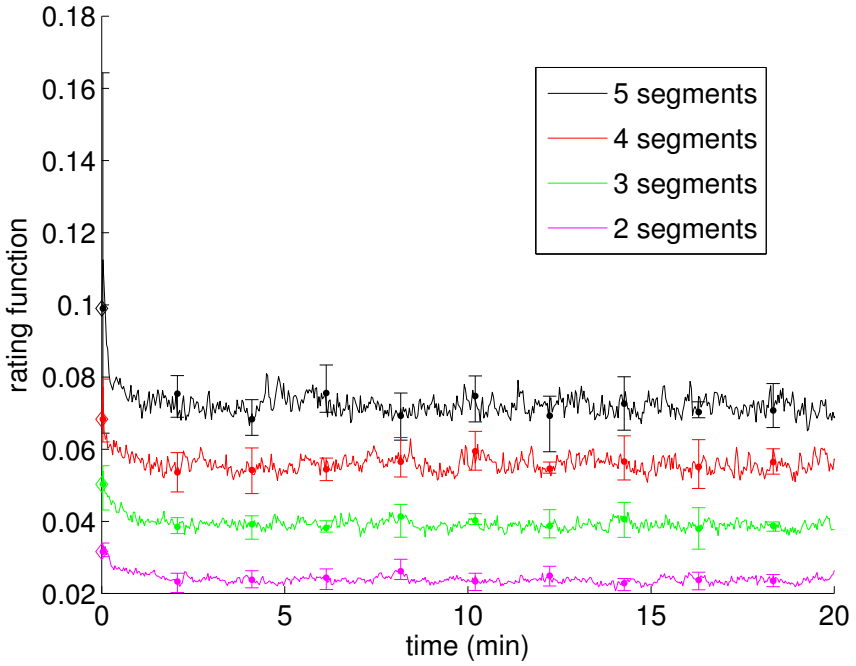


Figure 4.18: Rating function as a function of time for scenarios with 2 to 5 segments with 100% routed and 100% received.

priority value can be detected because all streams know when the first message should be sent. By simply waiting for the transmission of all priority IDs for a given amount of time (using a timeout), unused priorities can be identified and simply neglected.

In Algorithm 4.2, a detailed description of the initialization procedure is provided. The two parameters *minDist* and *waitTime* need to be chosen in advance. MinDist is the number of idle time slots between two scheduled messages. A time slot is defined similar to a time slot from Section 4.3.1. WaitTime is the number of idle time slots the algorithm has to wait until it can be assumed that the expected priority level is not sending and can be skipped. The algorithm counts idle time slots until minDist is reached. If the expected priority level equals to the streams priority, it starts sending. Otherwise, the following idle time slots are counted until waitTime is reached. For every time reaching waitTime, the priority level is increased. The example in Figure 4.19 clarifies the function of the algorithm and the meaning of the two constants.

If an ideal system is assumed, both constants, minDist and waitTime can be set to one. In an ideal system as represented by our simulations the nodes have zero computation time and are perfectly synchronous. If not mentioned otherwise, this setting is used for the experiments in this paper. In a real system the constants need to be increased. A more detailed discussion and reasoning is

Algorithm 4.2 Dynamic initialization algorithm

```
1: minDist = 1
2: waitTime = 1
3: priority_level = 0
4: min_dist_count = 0
5: wait_time_count = 0
6: for (each time slot) do
7:   if priority_level = priority on the bus then
8:     priority_level++
9:   end if
10:  if current slot is idle then
11:    if min_dist_count < minDist then
12:      min_dist_count++
13:    else if wait_time_count < waitTime then
14:      wait_time_count++
15:    else
16:      priority_level++
17:      wait_time_count=0
18:    end if
19:  else
20:    min_dist_count = 0
21:  end if
22:  if min_dist_count = minDist AND priority_level is this streams priority
     then
23:    start transmission
24:  end if
25: end for
```

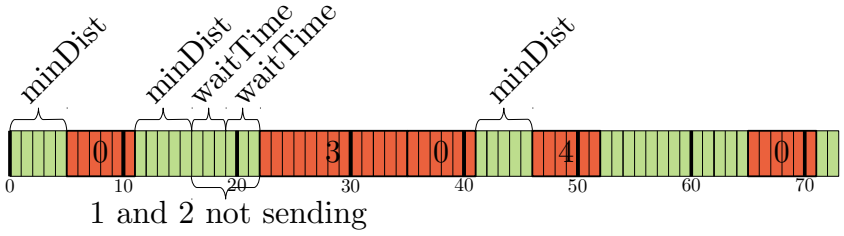


Figure 4.19: Example schedule of the dynamic initialization algorithm with minDist=5 and waitTime=3. The stream with priority zero has a period of 30 time slots. The streams with priority one and two are not present in the system and are therefore skipped after 11 idle time slots.

found in Section 4.6.2, where experimental results are presented. In conclusion, with the proposed approach the system can be fully automatically initialized. In contrast to a random initialization approach, the response times from the start are reduced. The algorithm does not depend on a simultaneous start of all streams, because in the worst case the stream is scheduled after one hyper period. This allows flexible activation and deactivation of streams.

4.6.2 Experimental Results

First, we compare different initialization phase strategies as shown in Figure 4.20. The worst case is when all offsets are zero and all streams start to transmit simultaneously. Although this is an overly unlikely scenario, it serves as a reference and indication of the upper bound. The second strategy is to initialize the offsets randomly. For this case, we made 1000 simulation runs with uniformly distributed offsets and show the maximum and mean value of these simulation runs. The last two cases represent our two proposed dynamic initialization strategies, one with decreasing and the other with increasing priorities. In this simulation, we assume all streams are active from the start.

Figure 4.20 shows that the dynamic initialization with decreasing priorities is always the best strategy. The dynamic initialization with increasing priorities performs worse than the maximum of the random initialization except for scenario hs_50a. This is caused by the fact that the priorities of the streams are mainly assigned according to their periods. If the streams with smaller periods are scheduled later, then they have releases at the beginning of the next hyper period. These releases can conflict with the earlier scheduled messages of the streams with larger periods. This is not the case if the streams are scheduled before their period, which happens more often when starting with the lowest priorities.

Next, we analyze the effects of the two constant parameters waitTime and minDist. A real system has inaccuracy that could lead to malfunction of the algorithm. For example, a stream could miss its slot because it didn't respond fast enough. The parameters need to be increased accordingly to avoid these problems. At the current state of development it is impossible to measure the effects on a real system, because the inaccuracy is highly dependent on the underlying hardware. However, we can estimate the effect by simulation with increased parameters.

Figure 4.21 presents the results of simulating the dynamic initialization for the scenario ls_50 and hs_90. For scenario ls_50, it shows that with increased waitTime the rating increases, but it is always better than the average of random initialization. Also, the parameter minDist has no influence. For the fully loaded scenario hs_90, the parameter minDist has some influence on the rating.

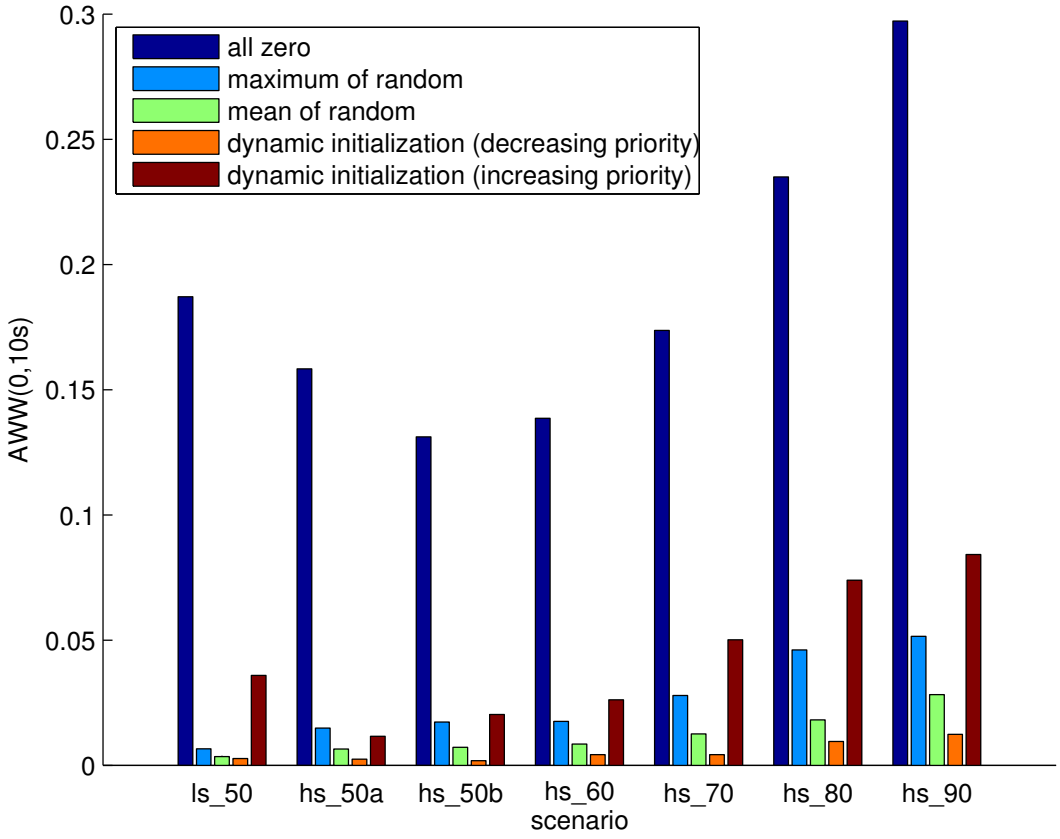
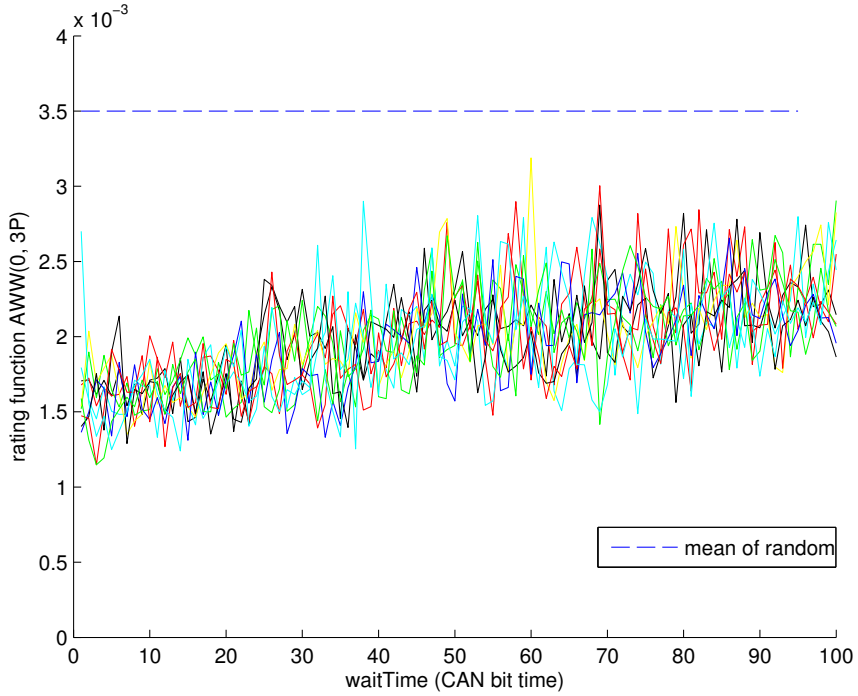


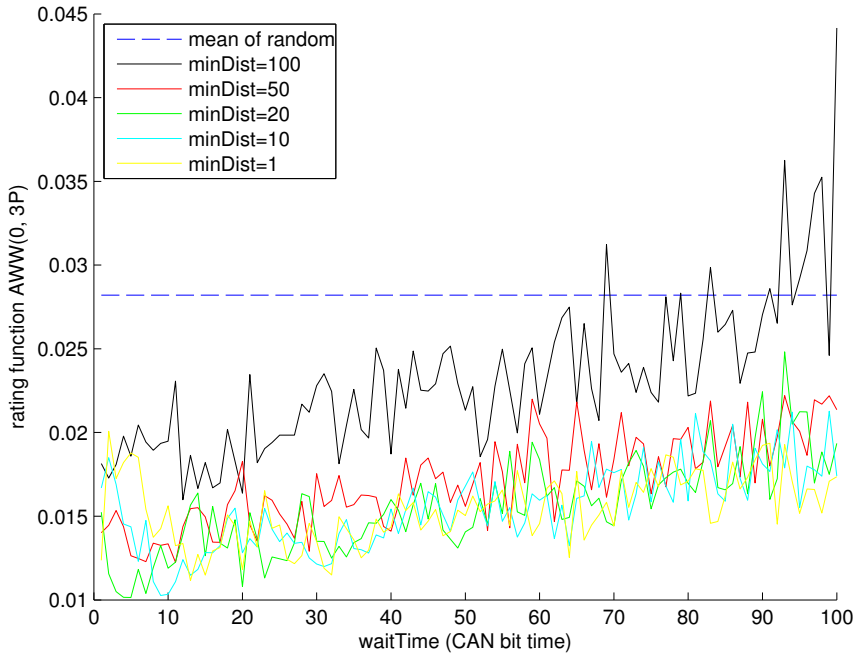
Figure 4.20: AWW for different initialization strategies.

However, only when both parameters are above 50, the rating is in the range of the mean random initialization.

The results of simulation of the initialization show the approach is robust in terms of variation of parameters. Further research on a real system needs to be done to confirm that the approach will work with the same results as in simulation.



(a) The different colors show the results for minDist varying from 1 to 90 in steps of 10 for scenario ls_50.



(b) The different colors show the results for different sizes of minDist for scenario hs_90.

Figure 4.21: Rating of the schedule after initializing all streams for different parameter settings. Both parameters are chosen in multiples of the time for one CAN bit (8 μs for ls_50, 2 μs for hs_90).

5

CAN+: Performance Enhancement by Modification of CAN

As already mentioned, the number of electronic components inside cars increase. New ECUs are not only dedicated to entertainment, but also for increasing safety. Advanced driver assistance systems, e.g., monitor the environment through various sensors, detect critical situations, and take proper actions or at least warn the driver. More and more mechanical connections are replaced by electronic ones to save energy and increase comfort, e.g., steer-by-wire.

All these electronic devices need a way of exchanging information on a fast, reliable, and robust way. Today, the most widespread communication system used in the automotive area is CAN [CAN91]. Using its priority-based event-triggered communication mechanism, it is appreciated for its properties of very short response times but still flexible design capability. The increasing number of electronic components due to the new applications boosts the workload of the bus. Besides the obvious fact that the data rate could be simply insufficient, an increase of workload leads to increased response times. In contrast to the approach in Chapter 4, where dynamic scheduling avoids concurrent bus access, in this chapter, we propose a solution where we avoid this problem by increasing the available data rate. One possible solution to increase the data rate is to add additional wiring harnesses. These additional buses, however, increase the costs and the weight of an automobile, which both is not desired. Another problem is that it is not always possible to split the electronic components to communicate on different types of buses, because complex gateways would have to be inserted for protocol conversion and routing, thus creating additional overhead and cost.

Our solution presented here to cope with the increasing demand for higher throughput is to increase the data rate on one CAN bus. Simply increasing the bit rate of CAN, however, is not possible, mainly because of the electromagnetic compatibility and hardware delays, but also because of used the medium access control (MAC). All the restrictions besides the MAC could be eliminated by using, for example, faster transceivers to reduce the delay or shielded cables to cope with electromagnetic interference. The MAC method nevertheless allows an increase of the bit rate during certain time intervals. The CAN+ protocol as introduced here analyzes and uses exactly these time slots in order to enhance data rate without disturbing existing CAN-conform hardware. Thus, our technology and methodology is backward-compatible. This allows to prolong the life of CAN for the next generation of distributed embedded systems.

This chapter is organized as follows. Section 5.1 extends the fundamentals of Section 2.2.2 and presents related work on increasing the data rate on CAN. Section 5.2 introduces the idea to insert additional information into data packets at controlled points of time in order to increase data rates. At the end of this section, the layout of the CAN+ protocol is defined. Section 5.3 describes the architecture and experimental setup that was used to verify and quantify the CAN+ protocol. In Section 5.4, the achievable speed increase is analyzed and the effects of the configuration of the standard CAN controller on CAN+ are described. The feasibility of the approach is furthermore shown by two case studies. Finally, electromagnetic compatibility tests strengthen the concept of CAN+.

5.1 Fundamentals and Related Work

In order to increase the data rate of CAN, we first need to understand why the data rate is limited. The CAN Specification 2.0 [CAN91] itself doesn't limit the maximum bit rate, as it only specifies that all nodes need to use the same bit rate. It is left for the physical layer to specify the maximum bit rate. There are three main physical properties that influence the maximum data rate and bus length: output resistance, propagation delay, and oscillator inaccuracy. Output resistance only limits the length of the bus, depending on the power used. Oscillator inaccuracy can cause errors when the clocks of two CAN nodes drift apart to much before synchronization. Due to the bit stuffing mechanism of CAN a synchronization is performed no later than after 5 CAN bits. As the clock accuracy of typical oscillators is in the order of 10^{-5}, this is not the limiting factor. Therefore, the propagation delay is the reason for the bit rate limitation. The delay consists of two components: Signal delay on the cable and delay within the controller and transceiver. This is the reason why the existing CAN protocol has the following limitations [Cor08]:

bit rate:	bus length:
1 Mbit/s	40 m
500 kbit/s	100 m
250 kbit/s	200 m
100 kbit/s	500 m
50 kbit/s	1000 m

The above limitations can be explained as follows: Errors occur, if during the arbitration phase a sending node cannot observe if it is overwritten until the end of the bit time t_{bit} as defined in Section 2.2.2.1. That means the propagation delay is not allowed to be longer than the time from the beginning of the bit time to the *sampling point*. The sampling point is the point in time, where the value of the bit is read. This gets clearer by looking at the example rate of 1 Mbit/s. Assuming the sampling point amounts to $\frac{2}{3}$ of the bit time (in this case 1 μs), the time from the start of one bit time to the sampling is about 700 ns long. Assuming moreover a transmission delay of 5 ns/m on the cable and a controller and transceiver delay of 200 ns, the total delay within 30 m is approximately $30\,\text{m} \cdot 5\,\text{ns/m} + 200\,\text{ns} = 350\,\text{ns}$.
[Cor08]

As can be seen in Figure 5.1, the worst case happens when node 2 synchronizes to the start bit of node 1. If the delay is too big, then node 1 reads the start of frame of node 2 as the first arbitration bit and would back off, even though it shouldn't at this point of arbitration. This simple example shows that the time between start of bit and sampling has to be at least two times the total delay, here 700 ns.

The delay of the controller could be minimized by using faster hardware, but the signal propagation delay on the bus is unavoidable. Therefore, the length of the bus significantly dictates the the speed limit.

In the previous paragraph, it was shown that a general increase of the bit rate is physically not possible. This speed limit is caused by the parts of the protocol, where several nodes need to access the bus simultaneously. This occurs twice during the transmission of one data frame: During the arbitration and acknowledgment phase. However, increasing data rates is possible in those time intervals where it is ensured, that only one node is sending. Then, the propagation delay of each bit is constant and all nodes receive the data with correct timing only delayed constantly.

The first idea of using CSMA with bitwise arbitration and different bit rates during different phases of the transmission was introduced by the µITRON bus [MMTS96]. The protocol was developed within the project TRON [Sak94]. However, there does not exist a detailed protocol specification and only simulations have been done.

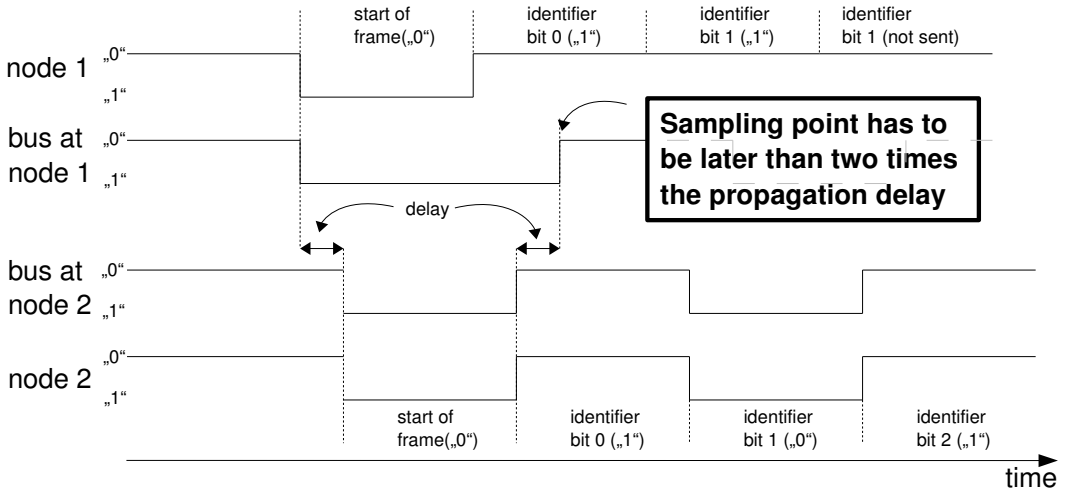


Figure 5.1: The impact of delay on the arbitration mechanism of CAN illustrated an example with two sending nodes.

The first to introduce the overclocking of CAN were Cena and Valenzano [CV99]. In their work, they present the idea of increasing the data rate in certain time intervals. To do so, they divided the transmission of a CAN message in two areas: areas, where multiple nodes can transmit (M-zones) and areas where the bus is reserved by a single node (S-zones) as shown in Figure 2.4. In the M-zones, the normal data rate is applied. In the S-zones, an increase in data rate is suggested. Cena and Valenzano arrive at the conclusion that only a speedup of four times the normal speed is possible. This limitation is reached, because the higher data rate is used to decrease the transmission time instead of increasing the amount of data per message. Therefore, only the transmission time of the S-zone is reduced and an upper bound is quickly reached, because the M-zone dictates the message transmission time. On the one hand, their new protocol idea decreases the response time of high priority messages, because a higher priority message, which is ready to transmit has to wait less time until the lower priority message is finished. On the other hand, not increasing the amount of usable data transmitted per message limits the maximum data throughput. This can be seen in Fig. 5.2 where the usable data rate of a transmission with constant packet length is compared to a transmission with increasing packet length.

Another problem with this approach is that it is not conform with the CAN specification according to [CAN91]. This means that using this protocol with normal CAN nodes is not possible, because the slow nodes will read errors and overwrite the bus with the dominant error messages. Cena and Valenzano suggest to use a reserved bit in the control field to flag those messages with

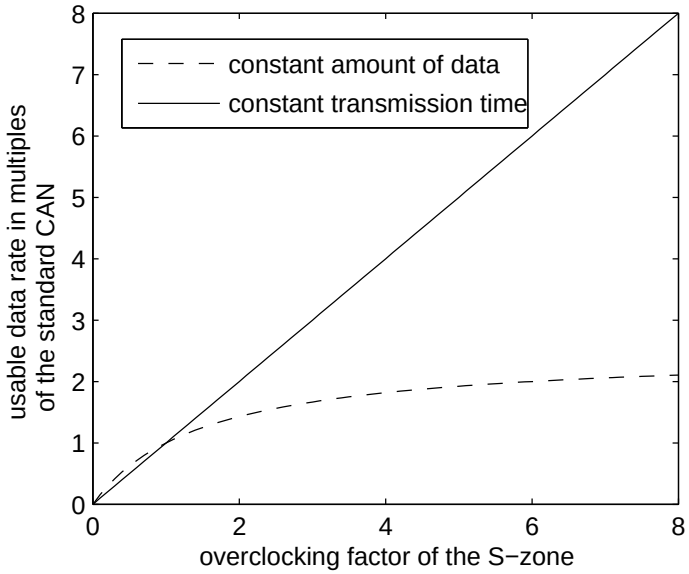


Figure 5.2: Increase of the usable data rate, when transmitting the bits in the S-zone with a different transmission speed than the M-zone. An overclocking factor less than one means decelerating the S-zone.

higher data rate. This modification still requires a redesign of the current CAN controllers. This is not practicable for a standard, that is more than 20 years on the market. Beside the already mentioned problems, it has to be noted that no practical experiments were made on achievable data rates.

Sheikh et al proposed more recently a protocol modification for CAN to increase the data rate and predictability [SHS10]. The method of increasing the data rate is similar to the principle introduced by Cena and Valenzano: During S-zone the bit rate is increased. In addition, the data field is extended by the so called *Data_Plus* field. This additional field allows to increase the data frame length to 1024 bytes, solving the problem of limited data throughput. Furthermore, the Data_Plus field is used to compensate eventually inserted stuff bits to achieve data frames with constant length that increases the predictability. An Field Programmable Gate Array (FPGA) prototype has been built that shows a reduction of average message transmission times of 50%. No results on maximal data throughput are provided. In [SSY10], an analysis of the performance of the modified protocol for real-time applications is presented showing message transmission jitter, and potential effects on the expected bit error rate. Despite all the advantages, the main disadvantage of this protocol modification still is that it is impossible to use nodes with the modified protocol and nodes with the standard CAN protocol on the same bus.

The relevance of increasing the data rate of CAN is emphasized by a recent proposal of Bosch, the originators of CAN. In their white paper, they suggest the protocol *CAN with Flexible Data-Rate* [CAN11]. It basically adopts the idea from Cena and Valenzano, by using two different data rates. Additionally, the maximum length of a data frame will be increased from 8 bytes to 64 bytes by using unused data length codes in the control field. The use of a higher bit rate will be indicated by sending a recessive bit during the first reserved bit in the control field. This should allow to integrate controllers with the new protocol into networks using the standard CAN. However, current CAN controllers expect a dominant bits at the reserved bits in the control field and the behavior in other cases is undefined.

Summarized, all the previously presented approaches don't allow to mix standard CAN nodes with nodes with increased data rate at the current state of development. This makes an acceptance of the protocols very difficult. In the following, we will propose a protocol that is fully backward compatible with the current standard CAN.

5.2 CAN+ Protocol

In this section, we propose an extension of the CAN protocol called CAN+ with which the data rate can be increased. Moreover, existing CAN hardware and devices not dedicated to these boosted data rates can still be used without interferences on communication. The major idea is a small change of the protocol. In particular, we exploit the fact that data could be sent in time slots, where standard CAN nodes don't listen. In particular, we first describe the time slots where additional data can be sent called gray zone. Then, we depict how we encode the data within the gray zone. Finally, the order in which data is transmitted is presented.

5.2.1 Usable Gray Zone

In order to understand the principle of the new protocol CAN+, a closer look at a single bit transmission has to be made. Figure 5.3 shows in the first row the transmission of a standard CAN bit time. A detailed description is provided in Section 2.2.2.1. We divide the bit time into three zones. The first one is the synchronization zone. In this zone standard CAN nodes expect an edge if a bit change happened. If edges in this zone are asserted at the wrong point of time, it will lead to an incorrect synchronization and subsequently to errors. The zone where the actual bit value is determined, is called sampling zone. In a perfect environment with perfect hardware, sampling is done at a certain point of the bit, but experiments show a significant inaccuracy of this sampling point.

These two zones limit the gray zone. During this zone, the bus can take any value without disturbing the standard CAN communication. This unused region is used by the new protocol by inserting information at a higher data rate. The newly inserted bits will be called *overclocked bits* in the following.

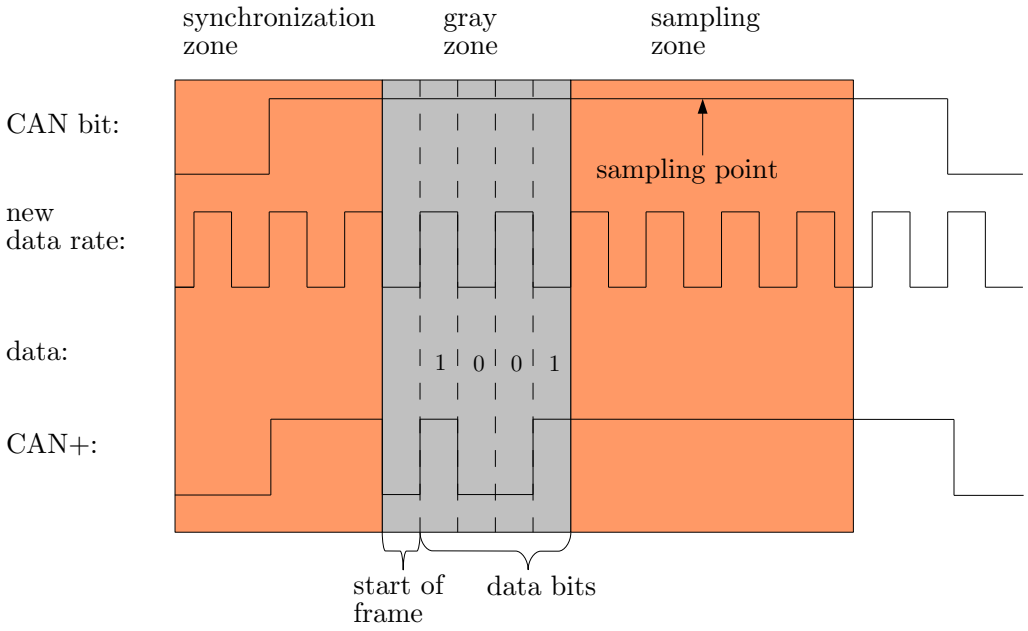


Figure 5.3: Transmission of a data bit for standard CAN and of four data bits for the CAN+ protocol.

5.2.2 Protocol within Gray Zone

Basically two approaches for transmitting the data within this small time slot are imaginable: Asynchronous or synchronous to the standard CAN bit. An asynchronous protocol such as the EIA232E [Ass91] commonly known as RS232 protocol would mean that a message within the gray zone would be transmitted. The start of the message is not exactly defined, therefore a signaling bit must be used. The synchronous approach would send the information relative to the synchronization edge of the CAN protocol. The implementation is difficult, because the time from the synchronization edge to the start of the gray zone is relatively long compared to the bit length of the overclocked bits. Therefore, an asynchronous protocol as shown in Figure 5.3 is chosen.

5.2.3 Layout of the CAN+ Protocol

As already described, the new protocol behaves, in principal, as the CAN protocol, except that it transports additional data bits in the data field. A normal CAN data message contains up to 8 bytes. The amount of the additional data depends on the *overclocking factor* ζ. The overclocking factor indicates how many overclocked bits are inserted during the transmission of one CAN bit. The maximum number of overclocked bits is evaluated in Section 5.4.1. Hence, an overclocking factor of $\zeta = 0$ means that the normal CAN protocol is used. The total maximum amount of transmitted data per message is:

$$N_{total} = 64 + 64 \cdot \zeta \text{ bits}$$

The additionally transmitted bits are ordered as follows. To make the protocol extension as easy as possible, a given data sequence of multiples of 8 bytes of data is divided into 64 bit vectors. Bit vector 0 contains the normal CAN data. Vector 1 to ζ denote the overclocked bits. They are ordered sequentially, meaning the first ζ bits of vector 1 are transmitted in the gray zone of the first standard CAN bit, the second ζ bits of vector 1 are transmitted in the gray zone of the second standard CAN bit and so on. If ζ is not a power of two it doesn't work out even and bits from consecutive vectors are transmitted within one gray zone. The principle is as demonstrated by an example with $\zeta = 4$ in Figure 5.4. This layout reduces the effort of reordering bits, because except for vector 0 the bits are transmitted in sequential order. Vector 0 is intentionally an exception, because it can be read by the standard CAN nodes.

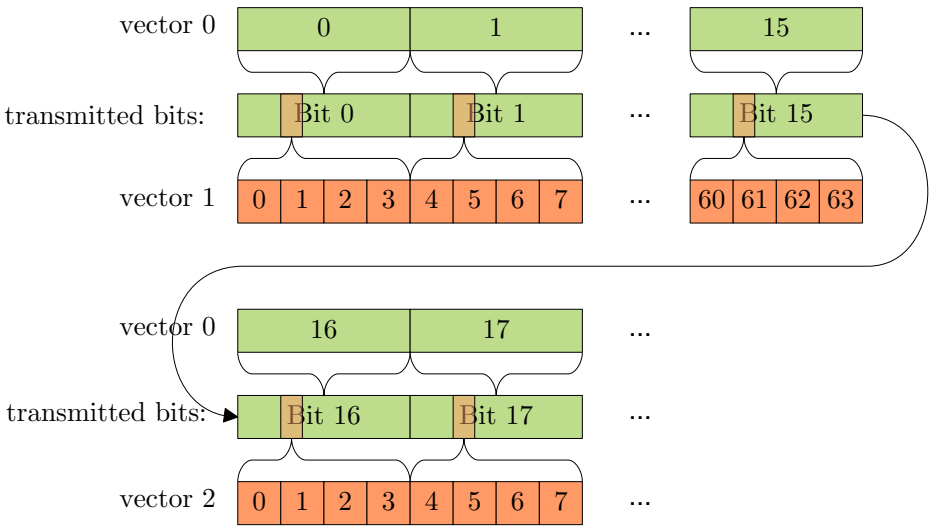


Figure 5.4: Order of inserted CAN+ data bits for $\zeta = 4$.

5.3 Hardware Architecture

In order to show the applicability and benefits of the CAN+ protocol, a hardware architecture for the CAN+ protocol has been designed. In this section, first, a CAN+ controller hardware module is presented. Then, an electric circuit realizing a transceiver needed for the increased data rates is given.

5.3.1 CAN+ Controller

The CAN+ module implements most of the CAN specification. In particular, it is responsible that the basic conditions for communicating with other CAN conform nodes are met. In addition to the CAN specification, the module implements the CAN+ technique of increasing the data rate backward compatible, as described in the previous section. A modular approach for fulfilling these complex requirements is chosen. The four modules can be seen in Figure 5.5.

The module *Interface* is used to connect the CAN+ controller to other modules. It is implemented by using asynchronous First-In First-Out queues (FIFO). At the current version, only messages with the size of eight data bytes are sup-

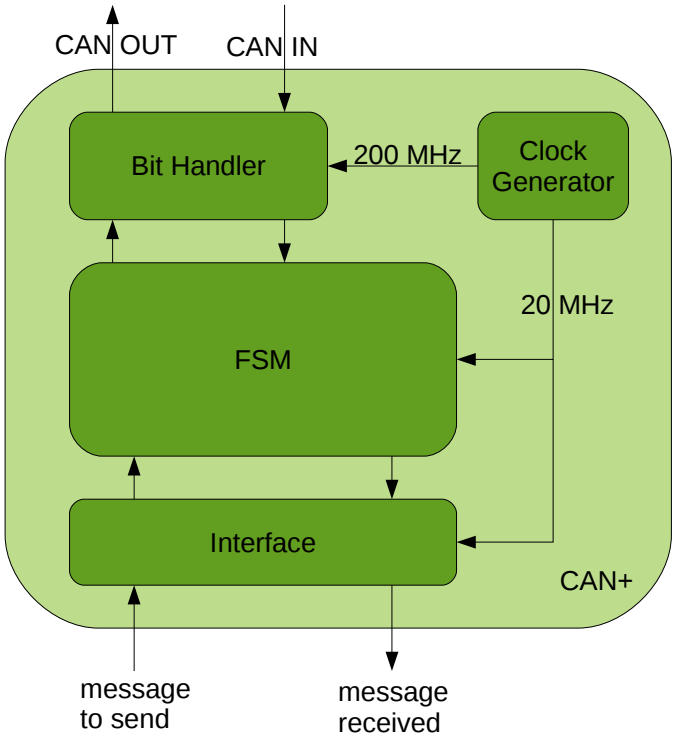


Figure 5.5: Schematic view of the hardware architecture of the CAN+ controller.

ported. This is the maximum size for standard CAN messages and allows the maximum throughput, because the overhead protocol data is constant. The identifier and the eight data bytes need to be written into the queue to send a message. The messages will then be transmitted as soon as the CAN bus allows to send it, i.e., the bus is not used by a node with higher priority. Reception works similar: As soon as a message is completely received, the identifier and the data bytes are written into the FIFO and can be read out by an external module. If the CAN+ controller is using an overclocking factor $\zeta > 0$, then $\zeta + 1$ vectors of eight byte need to be written into and read out of the FIFO for each message when receiving.

The module *Finite State Machine (FSM)* contains the control logic for the CAN+ controller. The main task is to comply eith the standard CAN protocol. The heart of the module are two finite state machines, which control when to send which bit. The finite state machines have the received bits as input and the bits to send as output. One final state machine has to send messages in a format conform to CAN, while checking if the send bits are transmitted correctly or if a node with higher priority is sending during arbitration. The other final state machine has to check if the received messages are correct, else an error message is transmitted. Also included in this module are the final state machines to store and load the data to and from the interface.

Two clock domains are used to have a very accurate sampling, while not stressing most of the design with the high clock rate. The module *Clock Generator* provides two clock signals that are synchronous to each other by dividing the higher clock signal to generate the lower clock signal. For a CAN+ controller with a standard CAN bit rate of 1 Mbit/s, 200 MHz and 20 MHz are generated.

The module *Bit Handler* is responsible for the reception and transmission of the single bits. This includes the synchronization as well as the sampling at the correct point. It transmits the read bit to the FSM module and writes on demand bits to the bus. The clock frequency of 200 MHz makes it possible to include additional information into a single bit.

5.3.2 CAN+ Transceiver

As the use of a standard CAN transceiver was not possible due to the high data rates, a new transceiver was designed which supports bit rates up to 60 Mbit/s. Figure 5.6 show the schematic of the design. The transceiver is implemented as a mountable board shown in Figure 5.7. It establishes the connection between the FPGA and the CAN bus. First, we will present the requirements and then show how they are solved. The results are presented in the last paragraph.

The requirements can be divided into two independent parts: transmitting and receiving shown in Figure 5.6. The signal CAN OUT is a digital signal, of voltage either 0 V or 3.3 V. It is responsible for the transmitting behavior. If

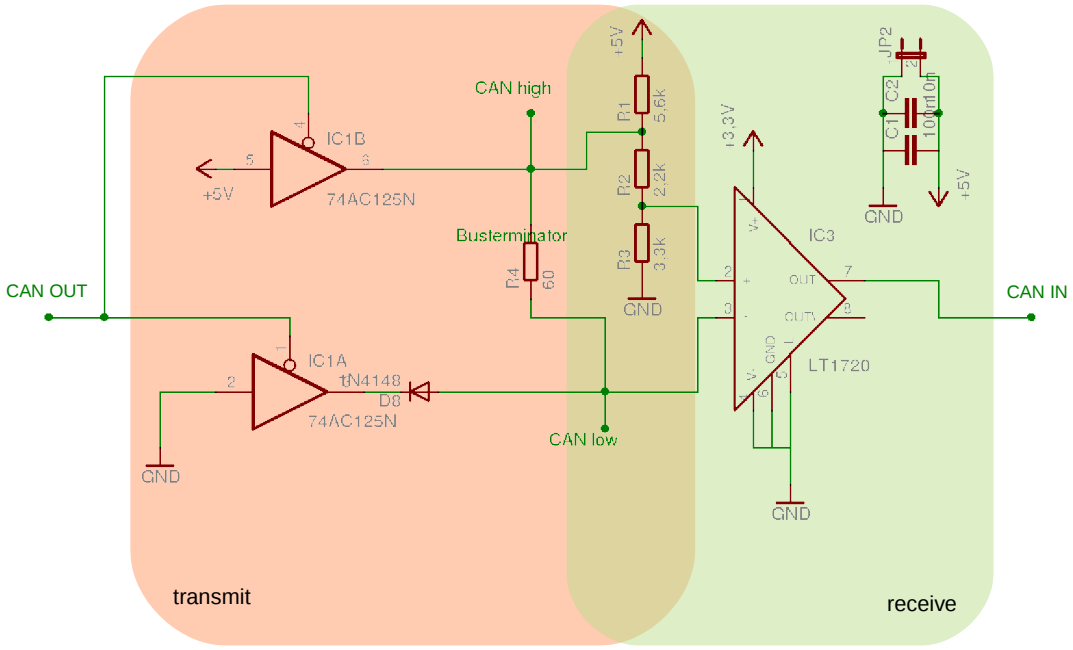
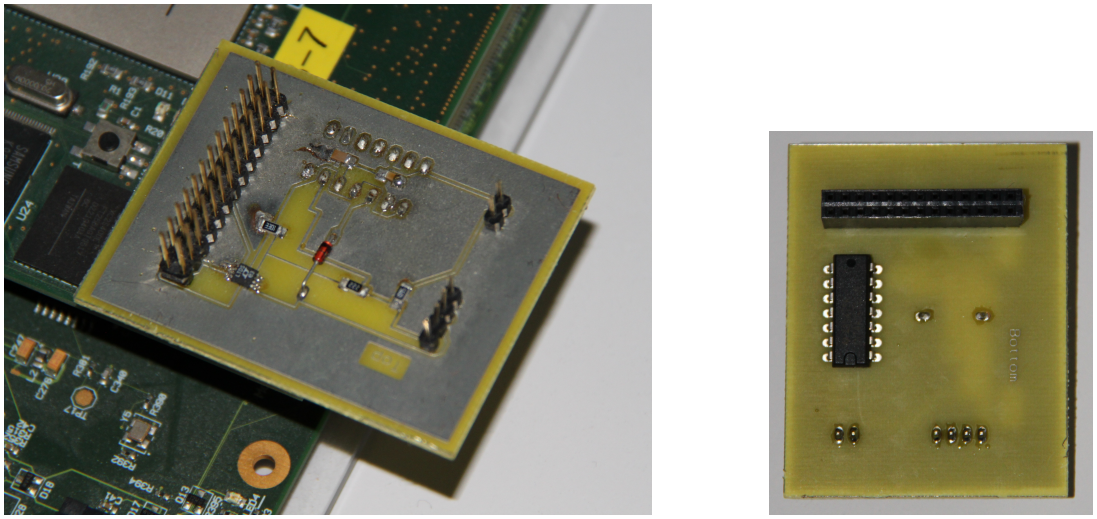


Figure 5.6: Schematic circuit of the CAN+ transceiver.



(a) top **(b)** bottom

Figure 5.7: Printed circuit board of the CAN+ transceiver.

the CAN+ controller sets CAN OUT to logical one, then the transceiver has to set the bus signal to one, which would mean applying 2.5 V to Bus High and Bus Low. If CAN OUT is set to logical zero the transceiver has to apply 3.5 V to Bus High and 1.5 V to Bus Low. The receiving part works similar, but into the other direction. The transceiver has to convert the signal on the bus lines to the digital signal CAN IN. These conversions have to be as fast and exact as possible, because it decides which data rate the transceiver is able to support.

The transmission requirement is solved by the following circuit. The needed 2.5 V are gained from the 5 V supply voltage by using a voltage divider. The CAN OUT signal is connected to two tri-state buffers (SN74AHCT125 [Tex03]). If the CAN+ Controller wants to send a logical one, the tri-state buffers disconnect and 2.5 V are applied to both bus lines. If a logical zero should be sent, then the tri-state buffers connect the additional voltages of 5 V and 1.2 V (limited by the diode). This pulls Bus High up and Bus Low down and achieves the desired voltage difference. The reception is done with a comparator (LT1720 [LIN98]). It is connected in such a way that as soon as a voltage difference of more than 1.2 V exists between Bus High and Bus Low a logical zero is applied to CAN IN, otherwise a logical one is applied.

In order to test the functionality of the transceiver, a clock signal with different frequencies was applied to CAN OUT and the results measured at the bus wires. The clock signal is simulating the transmission of a CAN sequence of alternating dominant and recessive bits. This test method leaves corner cases, but can still give a good approximation. Two examples of the resulting bus signals displayed with an oscilloscope are shown in Figure 5.8.

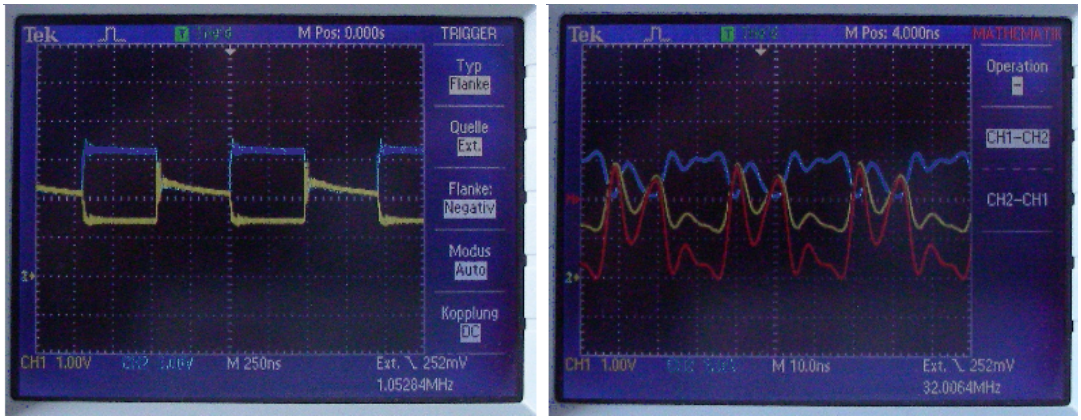


Figure 5.8: Output of the CAN transceiver with a clock signal of 1 MHz (left) and 32 MHz (right) connected to CAN OUT. The blue signal shows CAN high, the yellow signal shows CAN low. On the right, the red signal shows the difference between CAN low and CAN high.

5.3.3 Prototype Setup

The CAN+ controller is implemented as VHDL-module synthesizable on a Xilinx Virtex 2 FPGA mounted on an Erlangen Slot Machine (ESM) [MTAB07] Board. The build in digital clock manager [Xil04] is used to transform the available clock of 50 MHz into the suited 200 MHz sampling rate. A hardware prototype has been built to test the compatibility with standard CAN components and to measure the achievable data rates. The prototype consists of an FPGA for implementing the CAN-controller and an electric circuit as transceiver. The setup consists, as shown in Figure 5.9, of two FPGA boards connected each to a transceiver described in Section 5.3.2. The transceivers are connected to the CAN bus. As representative of a typical Controller Area Network, the CANcaseXL [Fal04] is connected to the bus. The CANcaseXL is used to evaluate if the CAN+ protocol is compatible to standard CAN controllers. For the physical data transmission, a simple flexible flat cable (FFC) with D-Sub connectors has been used. The 1 Mbit/s CAN is always used for comparison.

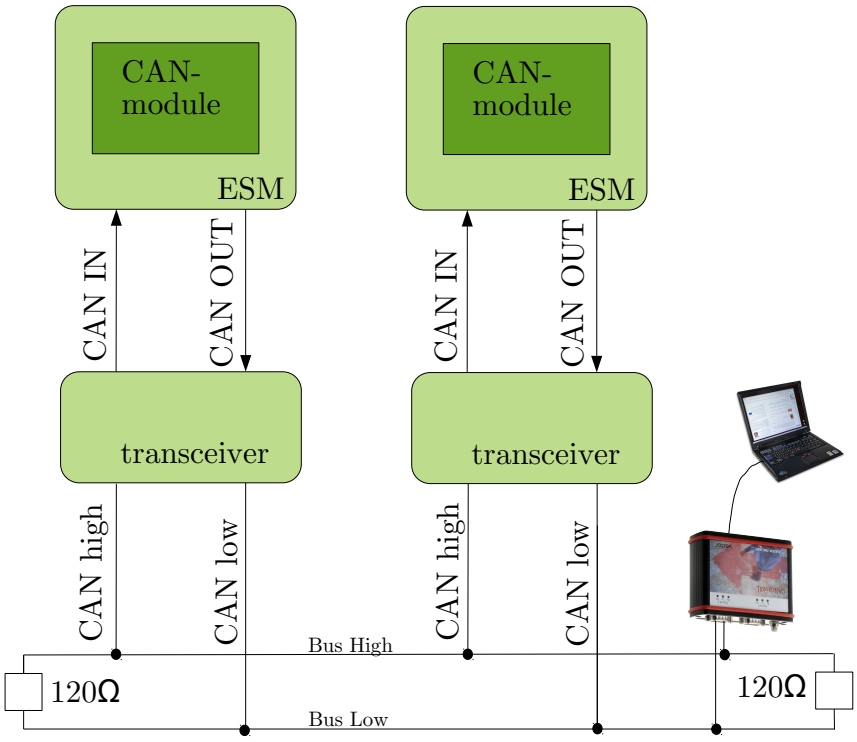


Figure 5.9: Schematic view of the prototype setup.

5.4 Experimental Results

In this section, we will first measure by a prototype setup which amount of overclocking is possible without disturbing the communication of standard CAN. Next, the performance of the CAN+ protocol is proven by demonstrating video transmission on CAN bus and the usability in the application domain of dynamic partial reconfiguration on FPGAs. To further show that CAN+ can be used in practical environments the electromagnetic compatibility is measured and compared to standard CAN.

5.4.1 Performance Measurements

Experiments with the setup depicted in Figure 5.9 were made to determine the length of the gray zone. To find out in which areas of the bit the CAN conform node isn't listening, an inverted signal has been inserted on different positions of the bit transmission interval, as shown in Figure 5.10. A stream including these corrupted messages is sent on the bus. The CANcaseXL, which uses the standard CAN controller Philips SJA1000 [NXP00], reads the messages. If the CANcaseXL doesn't signal any error messages, the controller is not noticing the inverted disturbance signal. The Philips SJA1000 has following configuration options:

- *Baud Rate Prescaler*

  Bit rate of the used CAN protocol

- *Time Segment 1*

  Time between start of bit and sampling point

- *Time Segment 2*

  Time between sampling point and end of bit

- *Synchronization Jump Width*

  Amount of adjustment per synchronization

- *Sampling*

  Single or triple sampling

In the following, we evaluate these parameters for influence on the length and position of the gray zone. The Baud Rate is set to 1 Mbit/s. The ratio between the time segments determines the position of the sampling point. It can be set to 50%, 62% or 75%. The usable gray zone for the three settings is shown in Figure 5.11. As described in Section 5.2.1, the synchronization zone and the sampling zone limit the gray zone.

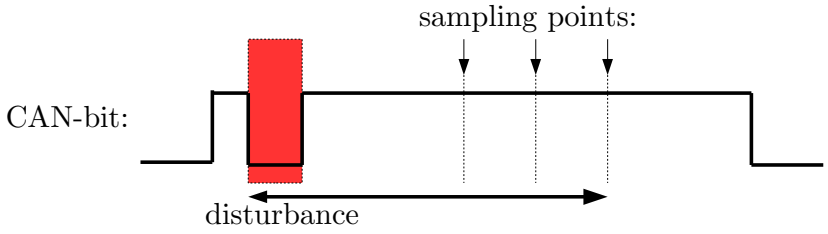


Figure 5.10: A CAN-bit with inverted signal is sent to size the grayzone by testing the acceptance of the corrupted messages.

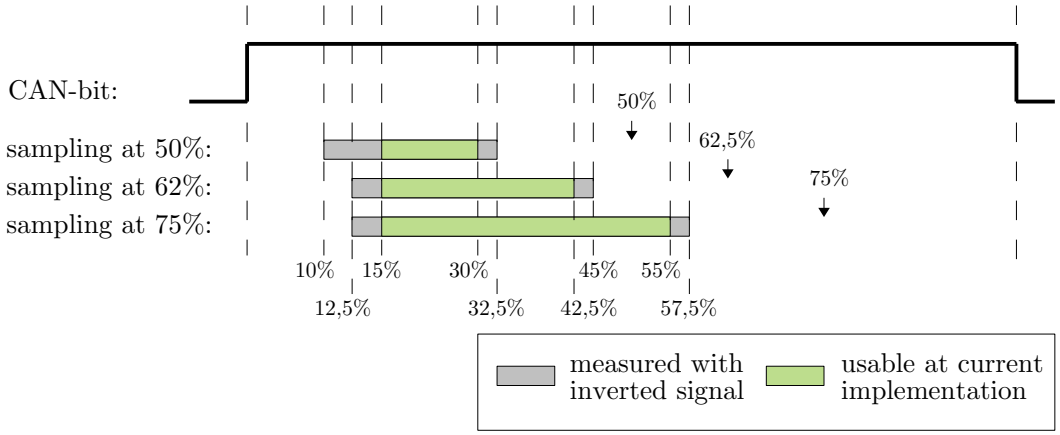


Figure 5.11: Analysis of the size of the gray zone with respect to the parameters Time Segment 1 and 2.

The beginning of the gray zone t_s, should be independent of the position of the sampling point. An exception is the beginning of the gray zone when using a sampling point at 50%, where the gray zone reaches into the synchronization zone. The controller synchronizes to the inverted signal if the normal CAN signal doesn't provide an edge. In consequence, the sampling zone is moved backward, as shown in Figure 5.12. When the sampling point is set at 50%, the sampling zone is still in the bit transmission interval. With a setting of 62% or 75%, the sampling zone is moved into the next bit transmission interval and the following bit is read instead which causes errors. The measurements show that this effect is boosted by a large *Synchronization Jump Width*. Apart from that the Synchronization Jump Width has no influence on the gray zone. The end of the gray zone t_e, i.e., the start of the sampling zone, is always 17.5% before the configured sampling point. This concludes that the sampling zone has a fixed size independent of the used parameters.

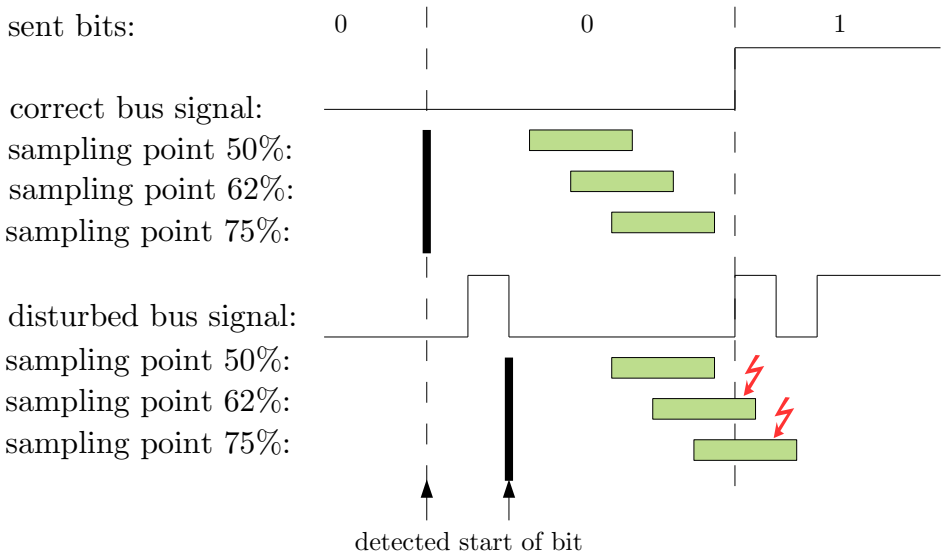


Figure 5.12: Effect of wrong synchronization caused by the inverted signal. The green highlighted sampling zone is shifted into the next bit.

The *Sampling* setting also does not have any effect on the results, because the two additional samples are taken with equal distance before and after the preset sampling point, as shown in Figure 5.13. If the sampling point lies on the inverted signal, at least one of the additional sampling points also lies on the inverted signal. Therefore, the majority reads the inverted signal.

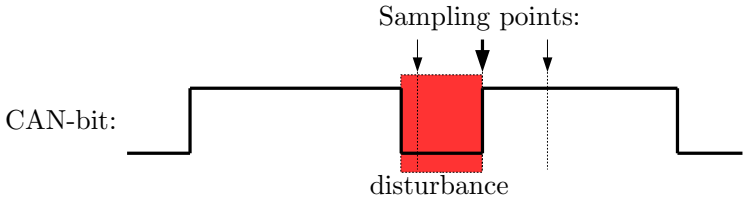


Figure 5.13: Triple sampling doesn't improve the size of the gray zone, because when the disturbance is at the middle sampling point at least another sampling point is affected.

In summary, the length of the gray zone ranges from 22,5% to 45% of the length of a bit transmission interval. However, only a part of the gray zone can currently be used for real data transmission, as shown in Figure 5.11. For our current implementation, there needs to be at least a delay of 15% from the start of the bit to the transmission of the high data rate bits. Additionally, when sending random data too close to the sampling zone, oscillations overshoot into the sampling zone.

This leaves a usable gray zone of about 15% to 40% of the length of a bit transmission interval. The bit length at 1 Mbit/s is 1 μs. So, the time for a data transmission ranges from 150 ns to 400 ns. With the current setup, a bit length of 25 ns can be used within the gray zone. Therefore, a maximum of $\frac{400\,ns}{25\,ns} = 16$ bit can be inserted. The used protocol for the gray zone has one bit overhead. This leads to an overclocking factor $\zeta = 15$ or, in other words, a usable transmission rate of 16 times the normal CAN.

5.4.2 Case study

In this section, two case studies are presented that demonstrate the features of the CAN+ protocol. First, using transmission of video on CAN shows the possible data rate of the approach. Then, the flexible overclocking adjustment is used to allow power saving.

5.4.2.1 Video Transmission

In order to show the functional correctness and robustness of the CAN+ protocol implementation and to verify the achievable transmission rate, a video application was implemented that makes it possible to send uncompressed video data over the CAN bus. The prototype setup as shown in Figure 5.9 is extended the following. A video stream taken from a webcam is fed into the first FPGA board. There, this video stream is packetized into CAN messages and sent over the bus. The second board receives and reconstructs the stream. With the available VGA-interface on the used FGPA board, the video is displayed on a monitor. Remarkably, videos from a webcam with 6,25 frames per second (fps) with a resolution of 320x240 in 8 bit gray scale may be transmitted without any errors with the presented CAN+ protocol, compared to less than 1 fps with standard CAN. Notable also that other CAN nodes may communicate without any interferences together with nodes using the presented overclocking scheme.

5.4.2.2 Dynamic Partial Reconfiguration

CAN+ was also tested in the context of dynamic partial reconfiguration (DPR). DPR describes the process of reconfiguring partial areas of an FPGA while the remaining part of the FPGA is still running. DPR may be used in combination with CAN+ to reduce power consumption and required area. The idea is to adapt the transmission data rate depending on the available communication bandwidth. This can be achieved by the following principle: Given that an application wants to transmit at a constant data rate. If the available bandwidth increases then there is more time to transmit the same amount of data.

Therefore, the transmission data rate can be reduce by using a communication module that requires less energy and area.

This concept was realized by developing partial CAN+ modules with different overclocking factors. Table 5.1 shows initial results gained by measuring the power consumption for the different CAN+ modules using the Xilinx Power Analyzer (XPA). The differences are not very significant and the overhead due to the reconfiguration itself doesn't allow an energy saving at the current implementation. Nevertheless, we see high saving potential: (1) The implementation independent quiescent power consumption will loose importance with further transistor miniaturization. (2) At the current stage, all modules use the same clock frequency. Modules with lower data rates could use lower clock frequencies that would in combination with voltage scaling further reduce the power consumption. (3) Newer FPGA generations allow to reconfigure in much less time and therefore reduce the overhead of reconfiguration.

In [ZT10a*], the scheduling for such modules for streaming applications is analyzed and first scheduling strategies suggested.

5.4.3 Electromagnetic Compatibility

The raising density of electronic components in cars opens new problems. All these components not only effect themselves, but also emit energy to their surroundings. This energy can lead to unwanted effects in other components. Therefore it is not possible only to consider the functionality of the component, rather the component should also not interfere other electrical components in the surrounding. This goal is summarized in the term electromagnetic compatibility (EMC). It can be subdivided into two categories: The first one is concerning the amount of energy the unit under test is allowed to emit and is called electromagnetic interference (EMI). In contrast, the amount of energy the unit under test can tolerate without malfunction is called electromagnetic susceptibility (EMS). In the following, we will investigate the EMC properties of CAN+.

The rest of this section is organized as follows. First, an introduction on EMC in general is given. Then, the test setup for the measurements is described.

ζ	0	64	159
dynamic power (mW)	78	142	234
quiescent power (mW)	337	337	337
total power (mW)	415	479	571

Table 5.1: Energy consumption for the CAN+ module using different number of overclocking bits ζ.

After that, using this setup the electromagnetic interference and susceptibility are tested and compared with standard CAN.

5.4.3.1 Introduction

In the field of EMC, one can distinguish between four major reasons for disturbance: Conductive coupling, capacitive coupling, inductive coupling, and radiative coupling. The first three reasons belong to the family of near distance disturbances. This means the interfering source is in the designer's hand and can be eliminated. This of course is not an easy task, still we don't see this as an unsolvable problem. In contrast to the near distance disturbances, radiative coupling is almost impossible to eliminate and to predict. For that reason, we focus our study on radiative coupling. In addition, the new protocol operates at higher frequencies so that radiative coupling might get a problem. Basic EMC literature [Goe97] says that radiative coupling needs to be considered when the length l_c of the interfering cable is comparable or exceeds the wavelength λ of the signal:

$$l_c \geq \lambda = \frac{c}{f},$$

where c is the speed of light and f is the used frequency. For CAN with 1 Mbit/s this can be calculated as follows:

$$\lambda = \frac{3 \cdot 10^8 \frac{m}{s}}{1\,MHz} = 300\,m$$

For CAN+ this can be calculated as follows:

$$\lambda = \frac{3 \cdot 10^8 \frac{m}{s}}{20\,MHz} = 15\,m$$

The calculation shows that on standard CAN radiative coupling is not of practical relevance as the length is restricted to 40 m. This also applies for slower variants. However, when using CAN+ it is necessary to investigate if it induces problems different from standard CAN.

5.4.3.2 Test Setup

To measure the electromagnetic emission and reception of CAN+ the prototype setup as shown in Figure 5.9 is used. We focused our measurement only on the cable of the setup for three reasons: First, the cable acts like an antenna and will therefore transmit and receive most of the disturbances. Second, the CAN+ nodes itself are much easier to protect because of their smaller size. Third, the prototype platform is far from being EMC-optimal designed and the complex functionality of the evaluation boards induces additional EMI effects.

Therefore, they would give unrealistic high results. We used two types of cables for our experiments to represent shielded and unshielded twisted pair wiring. As a shielded cable we used LEONI Dacar 533 FL09YBY 2x0,35+(0,35). It has the impedance of 100 Ω, which is not exactly suited for CAN, but shouldn't have any influence on the EMC-properties. A standard CAN 0,22 mm twisted pair copper cable was used to represent a simple unshielded cable.

5.4.3.3 Electromagnetic Interference

In this section, first the measurement procedure is described. Then the results of the experiments are shown and discussed.

Setup The first experiment for measuring the EMI of the cables that are communicating using CAN or CAN+ were carried out by placing the cable on top of a measurement table within an anechoic chamber. The transmitting and receiving CAN or CAN+ nodes had to be placed within the chamber. An antenna in 3 m distance to the cable was measuring the emitted radiation. Different measurements with connected and unconnected communication showed, that the emission of the evaluation boards was much higher than that of the cable. Shielding the nodes with a grounded copper plate didn't reduce the effect to a level where reasonable measurements could be made.

To be able to place the nodes outside of the measurement cell we moved our experiment setup to a so called GTEM cell. The GTEM cell is an extended version of the traditional TEM (Transverse Electro-Magnetic). In principle the TEM cell is a tapered coaxial line, from a coaxial feeding point with an air dielectric and a characteristic impedance of 50 Ω. In contrast to the TEM cell the GTEM cell, the coaxial line is terminated by a combination of discrete resistors and RF absorbers to achieve a broadband match. The outer conductor of this coax line is represented by the metal walls of the cell which provide screening for both the internal and external electromagnetic fields. The setup of the experiment is depicted in Figure 5.14. The cables of the prototype nodes are fed into the GTEM cell. The part of the cable which is measured is laid on a calibrated position on an insulator 5 cm above the hull orthogonal to the electric field. The measuring system is standardized to a standard anechoic chamber. The return of the cable was attached as close as possible to the hull to reduce disturbance. The ground wire of the unshielded cable was laid parallel to the communication cable. In this setup, only the horizontal flow of the cable could be measured. Now similar to the anechoic chamber, an EMI measuring receiver is connected to the GTEM cell. It is now possible to receive the EMI of the installed cable for different frequencies.

The settings of the EMI receiver are shown in Table 5.2, which represent typical values for EMC testing. The receiver always recorded the maximum

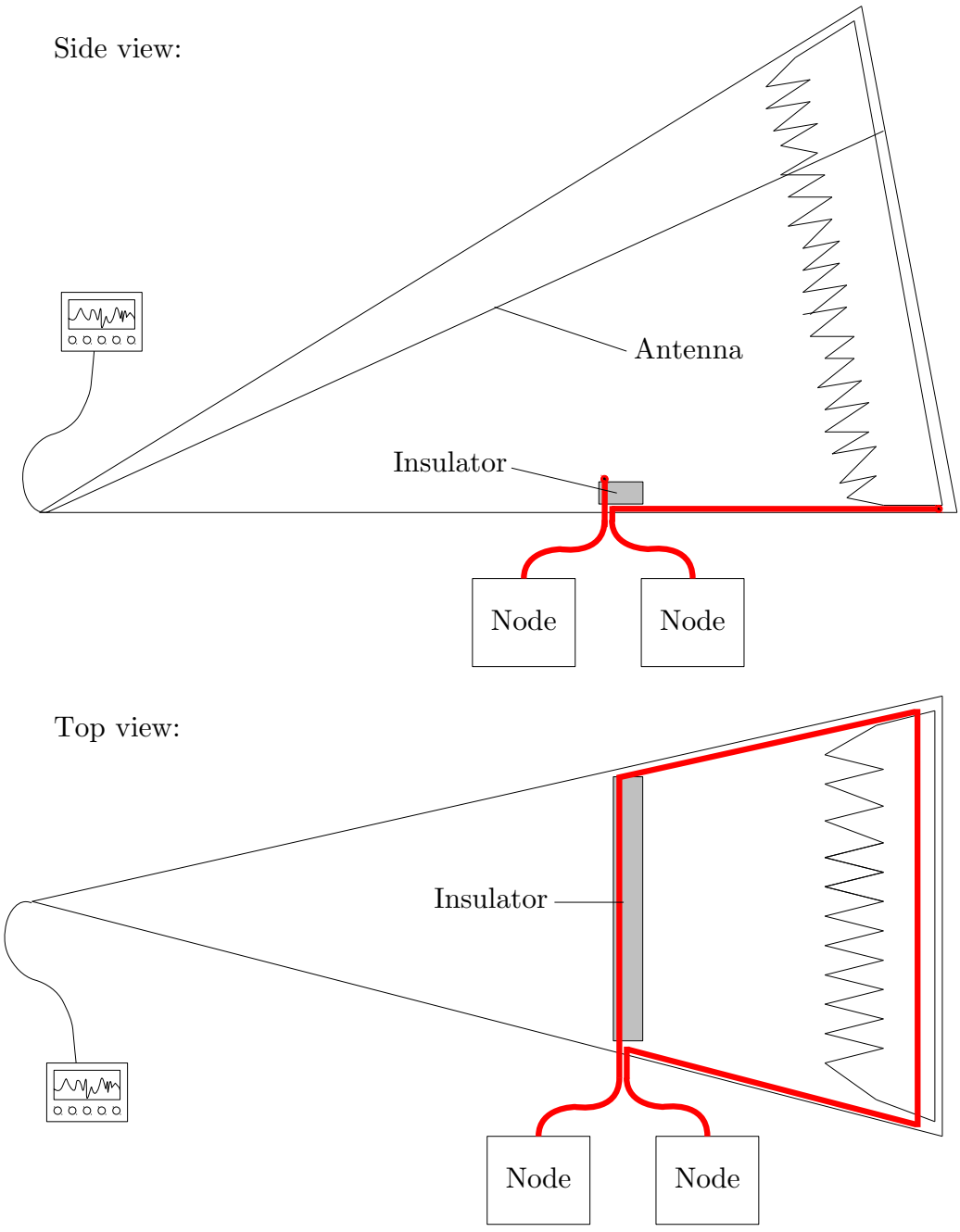


Figure 5.14: Setup for EMC measurement at a calibrated position in the GTEM cell.

values at each position. Several runs were made until the maximum values didn't increase anymore. The evaluation platforms turned out to be active amplifiers for noise in several frequencies, so we added ferrite beads to the cables to avoid the problem of transmitting noise into the cell.

Results With each cable the measurements were made for three different settings of the communication: (1) off: The communication is turned off. These results represent the noise that was transmitted into the cell. (2) CAN: The communication is using standard CAN. These results are used as reference. (3) CAN+: The communication is using CAN+ with 8 overclocked bits. Each overclocked bit is set on the bus for 25 ns. An example result for the unshielded cable at the calibrated position is shown in Figure 5.15. The blue plot shows the radiation while the communication is turned off. For most of the frequencies we could eliminate the noise and a comparison is possible.

The calibration to the standard anechoic chamber allows us to compare our results above 100 MHz to the DIN EN 55025 [DIN03]. Even thought below 100 MHz the conversion is not guaranteed to be exact, it can still give an approximation of the emission. For both cables and both protocols the results are below the prescriptive limits of class 5, which is the best class in this norm. The difference between the two protocols with the unshielded wire is shown in Figure 5.16. This shows us, that sometimes CAN and sometimes CAN+ gives lower results, but in average no trend can be seen.

The difference between the two test cables is shown in Figure 5.17. Here, again no trend can be detected. Therefore, simple unshielded cable can also be used for CAN+.

As the difference between the cables and the protocols at the calibrated position can not be measured, we did another series of measurements with the cables positioned closer to the measuring antenna. This has the effect that the overall signal strength is increased and therefore the difference between the different cables and protocols can better be seen. The setup of these experiments is depicted in Figure 5.18, the results in Figure 5.19. Now it is possible to distinguish between the different cables and communication ways. As expected we now can

Start frequency	30 MHz
Stop frequency	1 GHz
Step size	40 kHz
Resolution bandwidth	120 kHz
Measure time	50 µs

Table 5.2: Measurement settings for the CAN+ EMC tests.

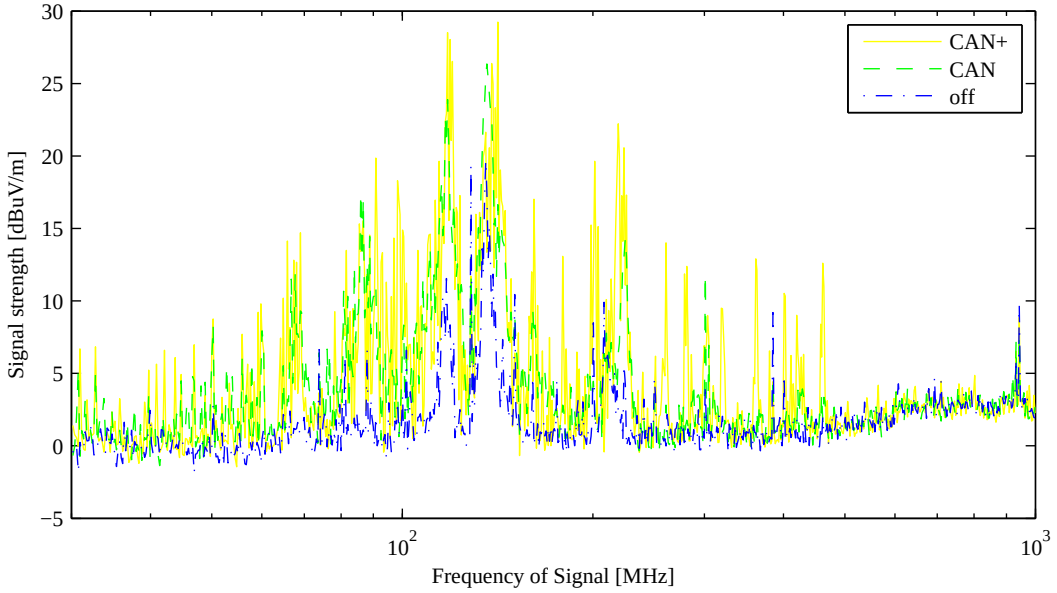


Figure 5.15: Signal strength at calibrated position in the GTEM cell for standard CAN, CAN+ and no communication using an unshielded cable.

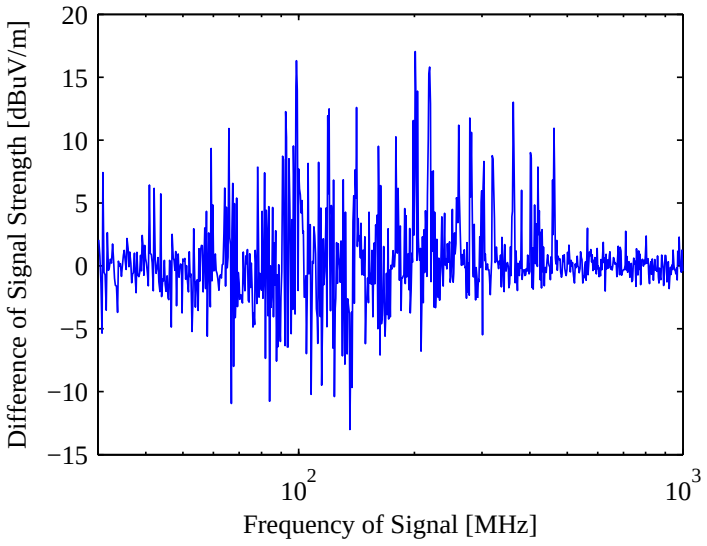


Figure 5.16: Difference of signal strength between CAN and CAN+ on the unshielded cable at the calibrated position.

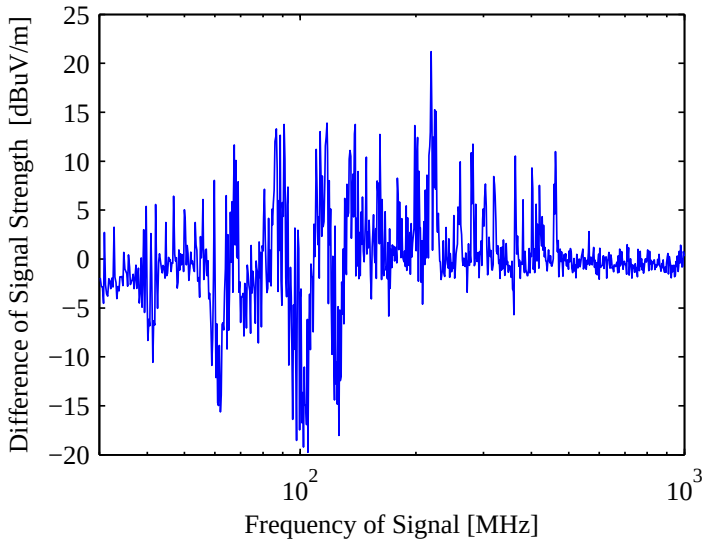


Figure 5.17: Difference of signal strength between the unshielded and the shielded cable at the calibrated position using CAN+.

see that the CAN+ protocol emits higher radiation than standard CAN. This effect is mainly discovered on the unshielded CAN cable.

However, on the shielded cable the effect is negligible. This behavior might origin in the optimization of the unshielded cable for CAN, whereas on the shielded cable both results are good. This suggests that it might be possible that a cable manufacturer can produce a cable that is optimized for CAN+ and will therefore reduce the radiation.

5.4.3.4 Electromagnetic Susceptibility

We also wanted to compare the electromagnetic immunity of CAN+ with that of standard CAN. This was done by reproducing the tests different car manufacturers impose to their suppliers. For confidentiality reasons we are not allowed to give detailed descriptions of these tests. We used the same setup as for the emission test, but used the antenna to transmit a disturbance signal on our unit under test. In our case the unshielded wire served as the unit under test. The wire was positioned as depicted in Figure 5.18. The car manufacturers demand a disturbance signal strength of up to 200 V/m. Unfortunately it was only possible to calibrate our amplifier to transmit at 150 V/m, even thought we used the setup where the CAN cable is very close to the transmitting medium. However, as we are only doing relative measurements this is not a problem.

The disturbance signal is a sinus wave of the adjusted frequency which is amplitude modulated by a sinus wave with the frequency of 1 kHz with the

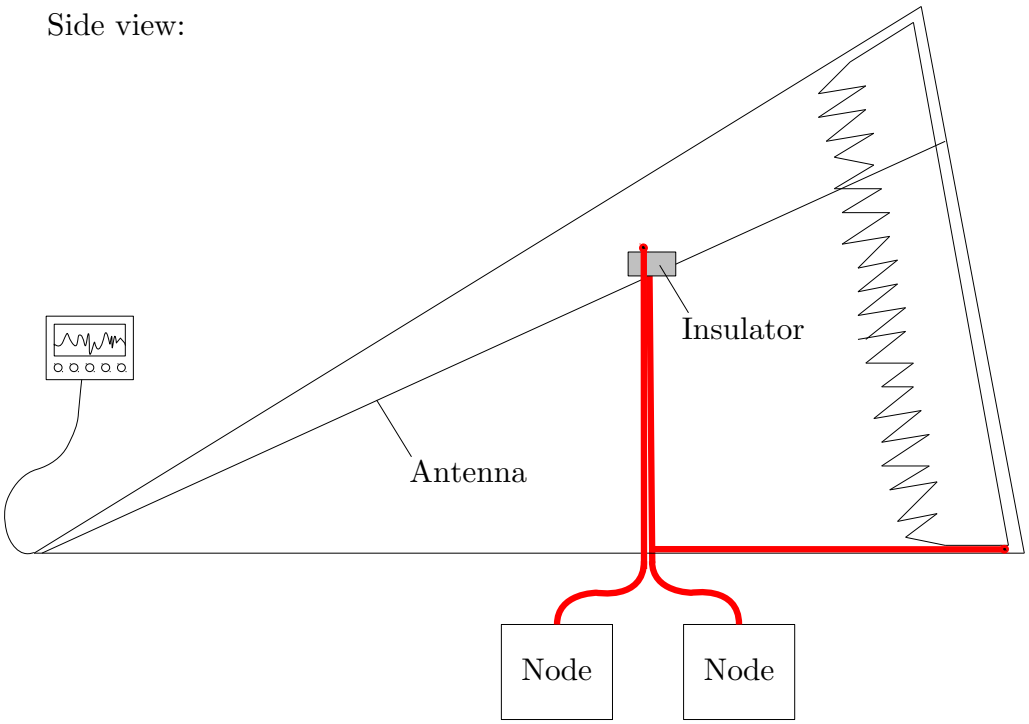
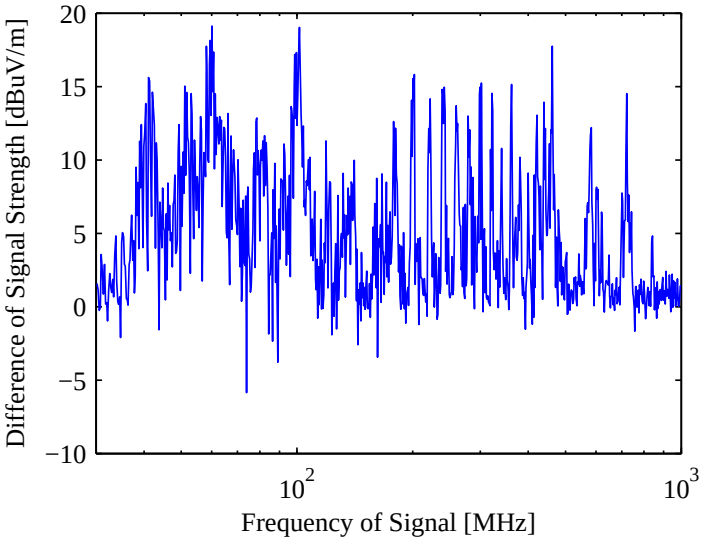


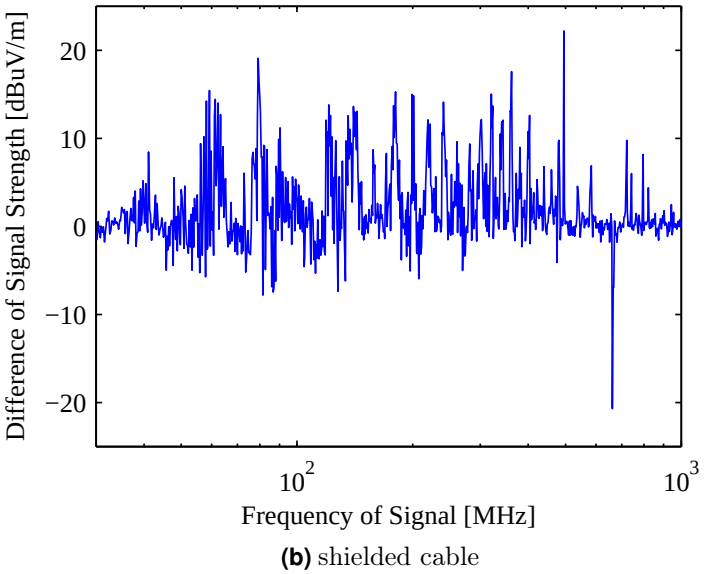
Figure 5.18: Setup for EMC measurement at close position to the antenna.

depth of 80%. The modulation is used, because this way real disturbances are better emulated. The frequency of the disturbance signal ranged from 100 MHz to 1 GHz. Again, the two prototype boards were connected to the bus wire and communicated by sending 1111 messages per second. The errors that occurred were measured in two ways. First, the standard CAN error messages, and second, the data errors were counted. The standard CAN error messages could be easily counted, since they are specified in the standard. The sending node dropped the message, when an error occurred during transmission. A data error occurs when the transmitted data and the received data are not the same. In standard CAN, data errors are protected by a checksum. Therefore, it is almost impossible to produce a data error without triggering an error message. At the moment, CAN+ is not protecting the overclocked bits. So, it is possible to get a data error without recognizing it. The data errors can be counted at the receiver, because we are sending a counter that is incremented by one every message. Therefore, the receiver knows what it should receive and can compare it to the actually received data.

The results of the immunity test are shown in Figure 5.20. While testing the standard CAN, only CAN errors occurred. In contrast to that during the CAN+ tests, above a disturbance frequency of 140 MHz only data errors occurred.



(a) unshielded cable



(b) shielded cable

Figure 5.19: Difference of signal strength between CAN and CAN+ at the close antenna position.

Below 140 MHz much more CAN errors occurred then data errors, but some data errors were caused by CAN errors. So, it was impossible to distinguish between data errors that could be traced back to CAN errors and errors that only appeared in additional CAN+ bits.

Comparing the two protocols in their susceptibility behavior, we have to divide the results of Figure 5.20 into three regions: below 140 MHz, between 140 MHz and 170 MHz, and above 170 MHz. Below 140 MHz both protocols are heavily influenced. On CAN+ the disturbance signal is causing roughly twice as much CAN errors as on CAN. This behavior is difficult to explain, since errors in the overclocked bits don't cause CAN errors. However, we can imagine that the influence of the disturbance signal could cause a time displacement of the overclocked bits, which in consequence leads to CAN errors. A measurement of the data errors in that region is meaningless because the CAN errors cause an interruption of the transmission. Between 140 MHz and 170 MHz the CAN protocol is not influenced at all and only the overclocked bits of CAN+ are influenced. The results here show that the maximum number of data errors per second is 50. This number is really small compared to the amount of data that is additionally transmitted. Above 170 MHz neither CAN nor CAN+ show any errors, except at 288 MHz. There, the standard CAN protocol showed more errors then CAN+. This behavior must be very specific to our setup and some resonance frequencies of the prototype board might be hit.

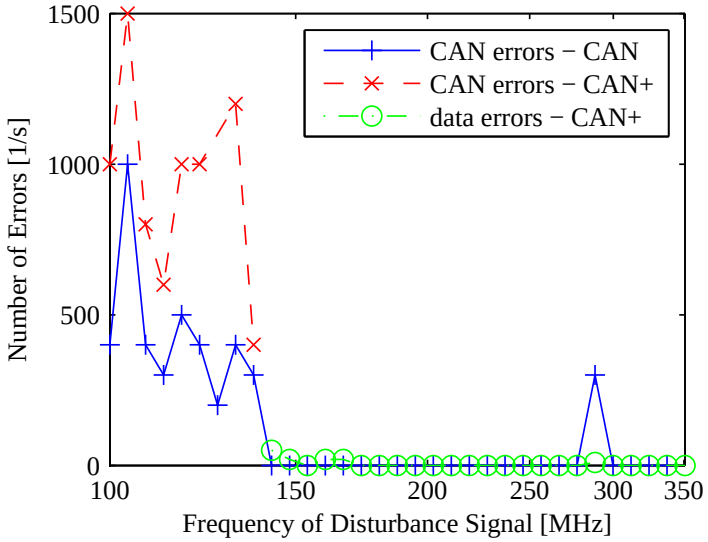


Figure 5.20: Number of errors that occurred when irradiating the communication with different frequencies.

In summary, the results of the electromagnetic susceptibility test showed that the new CAN+ protocol is more sensible, but the amount is very low compared to the amount of speedup.

6

Evaluation and Validation on a Prototype System

Designers of distributed embedded control systems face many design challenges related to change of system configuration, functionality, and number of participating computing nodes, which affect the usage of the communication bus. The concept of self-adaptivity of participating nodes plays an important role in reducing design effort while guaranteeing high system performance. The afore described methods have proven their advantage in simulation. However, it is still necessary to test the algorithm in a real physical environment. In this chapter, we use FPGAs with their capability of performing rapid system prototyping. This allows to achieve the following aspects:

1. validate simulation,
2. estimate computing time,
3. determine algorithm parameter and
4. make the results visible.

This chapter is organized as follows. Section 6.1 describes the architecture and experimental setup that is used for the evaluation and demonstration of the developed methods. Section 6.2 presents the integration of video streaming into the prototype system and the application of PPLA for video streaming. Section 6.3 proposes a testbed to analyze the response time of messages. This

testbed is used to evaluate different implementation options of DynOAA. Finally, in Section 6.4 the prototype system is used to investigate the use of CAN+ with a lower standard CAN bit rate.

6.1 Prototype System Setup

In this section, we describe the prototype system that is used to show the combination of the methods described in this thesis. The goal was to build a prototype that is as simple as possible but still represents the characteristics of a typical automotive system. We chose FPGAs as prototyping platform because, FPGAs allow software-only implementations using a soft core processor, pure hardware solutions, or solutions that combine hardware and software to achieve the same functionality on a single platform. Moreover, in foreseeable future it is expected that FPGAs are utilized in automotive ECUs of the final system [GNW+06, GGCG10], too.

A schematic of the prototype system is shown in Figure 6.1. Four FPGA boards are connected by a standard two wire CAN bus: Three *Atlys boards* and one *DE2-115 board*.

1. The main processing unit of the Atlys board is a *Spartan-6 LX45* FPGA. Besides several other peripherals, the Atlys board offers the possibility of showing HDMI video. Furthermore, the stereo camera module *VmodCAM* can be connected. It is displayed on the top left and top right corner of Figure 6.1. The Atlys boards are connected to the CAN bus with a CAN+ transceiver as shown in Figure 5.6 on page 109.

2. The DE2-115 board offers a *Cyclone EP4CE115* FPGA, which we mainly use to implement a *Nios 2* soft core processor. For the physical connection between the DE2-115 board and the CAN bus, we used a standard 8-pin CAN transceiver [LIN01]. We developed a simple PCB that is plugged on the I/O pins of the board. It is connected by two standard I/O pins for receiving and transmitting the CAN bus signal.

It has to be noted, that it is necessary to share a common ground which is established by using a third wire within the CAN bus cable. Sometimes more than one CAN controller are implemented on one FPGA. This is solved by using a logical internal bus. The CAN OUT signals are combined by a logical and to one CAN OUT signal that is connected the transceiver. The CAN IN signal from the transceiver can be connected to all CAN IN ports of all CAN controller.

The prototype system offers several sensors and actuators to visualize the effects of the proposed methods. On two Atlys boards there are signal lights that can be triggered by CAN messages. Two VmodCAM modules provide

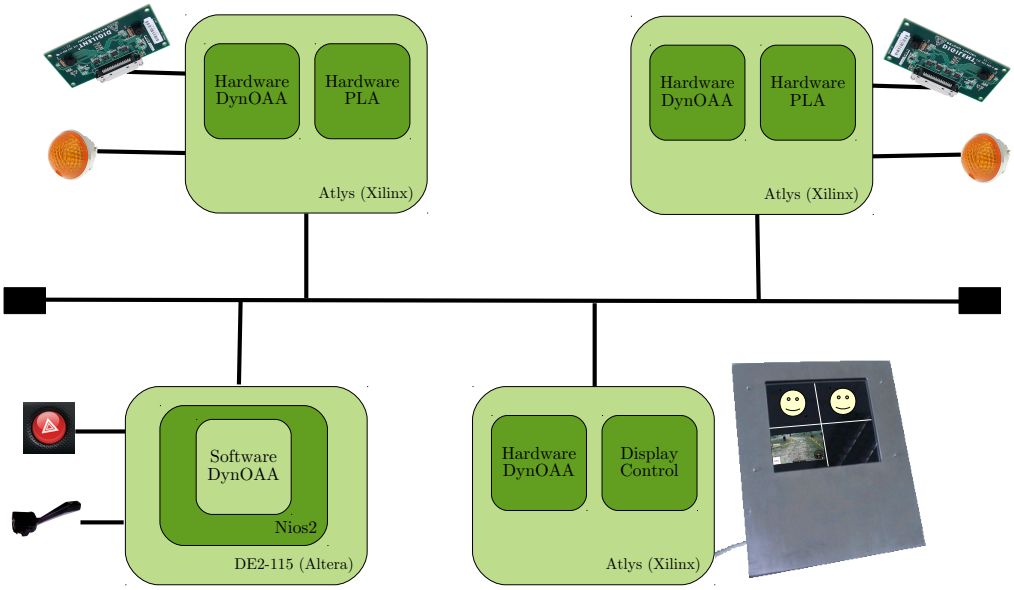


Figure 6.1: Schematic of the prototype system showing all actuators and sensors.

four video streams that can be displayed by a monitor. The details of the transmission of the video streams are explained in Section 6.2.1. Switches on the boards can be used to activate the following modes that represent all aspects of this thesis:

CAN+: One camera streams a full screen video over CAN+ to the monitor.

PPLA: The four cameras stream each a quarter screen video over CAN+. They can be individually turned on or off. Additionally, it is possible to turn PPLA on or off.

DynOAA: The four nodes generate CAN traffic typical for an automotive bus described in Section 4.4.1 on page 75. The workload can be increased and decreased by two pushbuttons. DynOAA can be turned globally on or off. It is also possible to reset all offsets to generate the worst case situation. The AWW is displayed to show the performance.

Mixed: Video streams and CAN traffic are transmitted at the same time to show the compatibility of the approaches.

6.2 Fair Bandwidth Distribution

In Chapter 3, we developed distributed algorithms to allow fair bandwidth distribution. The most advanced algorithm, the PPLA, was evaluated on a CAN simulation. In the following, we want to gather real performance numbers by the example application of video streaming over CAN. We choose video streaming because it perfectly fits our model of bandwidth streams and allows visual demonstration of the developed approach. The user scenario is the following: There exist four cameras that can stream their video on the CAN bus. They are able to stream with two different resolutions. A high resolution where a video with 16 fps can be transmitted if the full bandwidth of the CAN bus is used, and a low resolution where a video with 16 fps can be transmitted with one fourth of the bandwidth. The user can now choose the number and quality of the streams. The goal is to show the adaptability of the organic system.

6.2.1 Integration into the prototype system

In order to integrate bandwidth sharing into the prototype system, we first need to enable video streaming over CAN. We use CAN+ to achieve an acceptable data rate of around 1 Mbyte/s. Further details on the CAN+ implementation are provided in Section 6.4 on page 152. As described in Section 6.1 each Atlys board has two cameras connected and a reference design allows to stream videos with a resolution of 1600x900 with 25 fps. When using 8 bit per pixel, the full transmission would require a data rate of 36 Mbyte/s. We solve this conflict by streaming the video with a resolution of 400x225 that offers still a reasonable quality on the small prototype monitor.

Figure 6.2 shows the architecture on the Atlys boards, where board 1 represents a transmitter and board 2 a receiver. First, we describe the transmission of one 400x225 video stream, i.e., high resolution. Then the design is extended to support multiple 200x112 video streams, i.e., low resolution, and fair bandwidth sharing. The interface between the reference design and the video transmission is realized by a true dual port memory, the *Picture buffer*, implemented as embedded ram on the FPGA.

On board 1 in Figure 6.2, the reference design is modified to write every forth pixel in x and y direction of the pixel stream into the memory Picture buffer. The *Interface FSM* reads from the memory as soon as the *CAN+* module signals a free bus. The address of the memory is 17 bits wide. We use the standard CAN bits of the CAN+ message to transmit the memory address of the pixels whereas the pixel values itself are transmitted in the overclocked bits. This is done by using the 17 least significant bits of the 64 standard CAN bits to transmit the memory location of the first pixel of the message. The addresses of the following pixels are in ascending order. On board 2, the Interface FSM

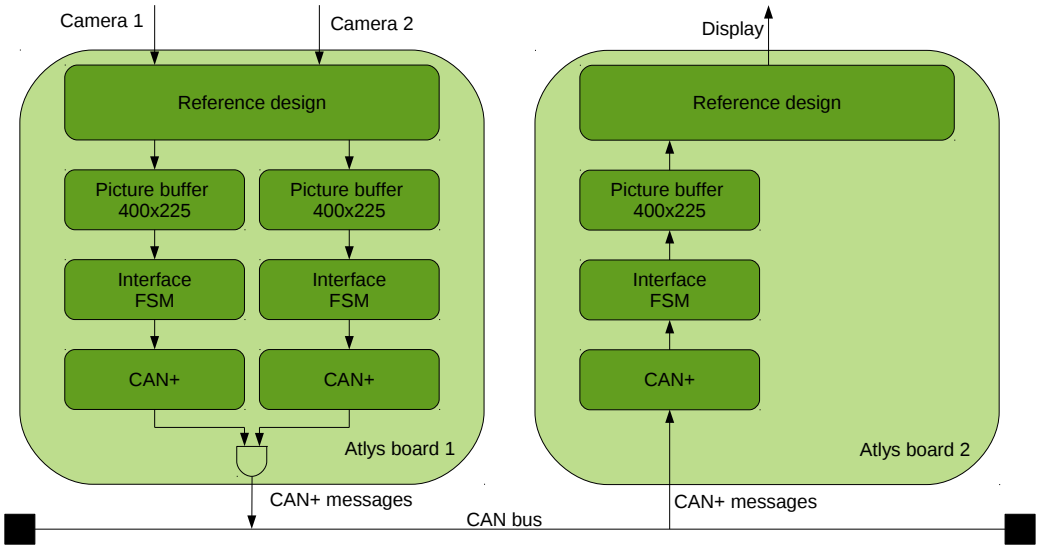


Figure 6.2: Architecture of the video streaming for demonstrating bandwidth sharing using the reference design for two Atlys boards.

writes the pixels from the received CAN+ messages at the transmitted address. The reference design is modified to display every pixel from the Picture buffer as a 4x4 block. The resulting behavior is that the video picture is scaled to the full display size.

In order to transmit with low resolution the Interface FSM reads only every second pixel in x and y direction from the Picture buffer and sends the pixel in ascending order. This results in a pixel order in the Picture buffer of the receiving board as shown in Figure 6.3. The receiving board stores up to four streams by simply offsetting the write address with multiples of 22400 depending on the CAN identifier. A split screen showing four streams at the same time is now realized by displaying the top quarter of the Picture buffer in the top left quarter of the display, the second quarter of the Picture buffer in the top right quarter of the display and so on.

The integration of fair bandwidth sharing was achieved by using the PPLA as admission control in the Interface FSM. The modified version of the PPLA is used similar to Section 3.5.2 where the adaptation is multiplied by the length of transmission of one CAN message l. The modified PPLA as described in Algorithm 3.2 with the modification of Section 3.5.2 requires transformation to be efficiently implemented in hardware. There are two divisions in line 1 and line 2 and one multiplication in line 8 that should be avoided as they require a significant amount of resources. The variable *load* is only used in line 3 which we can reformulate the following:

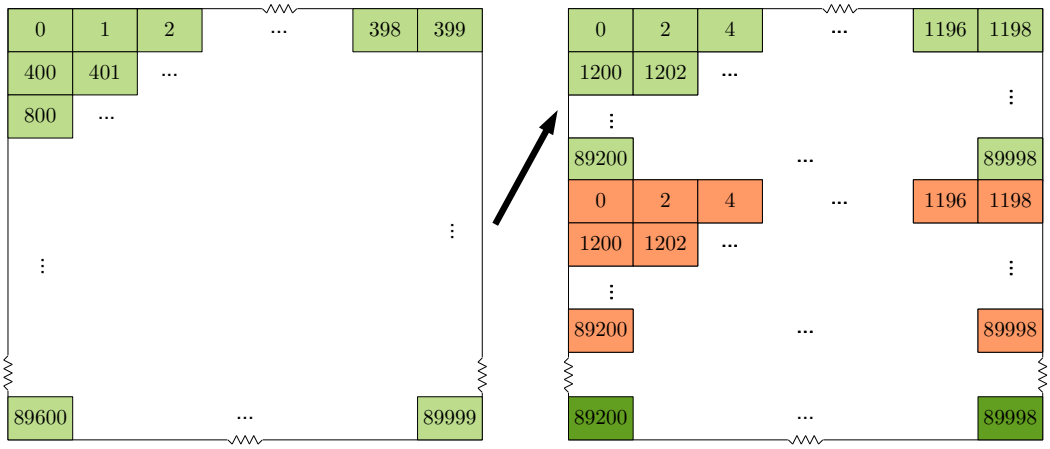


Figure 6.3: Pixel reordering when converting from 400x225 to 200x112.

$$\begin{aligned} load &> 1-\epsilon \\ \frac{m_{sent}}{l_{mi}} &> 1-\epsilon \\ m_{sent} &> (1-\epsilon)\cdot l_{mi} \end{aligned} \tag{6.1}$$

Algorithm 6.1 Adjustment of T_i for the modified PPLA during one learning step

1: $load = \frac{m_{sent}}{l_{mi}}$;
2: $success = \frac{nSuccess}{l_{mi}}$;
3: **if** $(load > 1-\epsilon)$ **then**
4: $\quad \Delta = success$
5: **else**
6: $\quad \Delta = -(1-success)$
7: **end if**
8: $T_i^{(t+1)} = T_i^{(t)} + \eta \cdot \Delta \cdot l$;

The right side of Equation (6.1) is constant and now only a simple comparison is necessary. The remainder of the algorithm can be reduced to two equations by solving line 8 depending on the result of the if-clause:

1. $T_i \Leftarrow T_i + \eta \cdot \frac{nSuccess}{l_{mi}} \cdot l = T_i + \frac{\eta \cdot l}{l_{mi}} \cdot nSuccess$

2. $T_i \Leftarrow T_i + \eta \cdot (\frac{nSuccess}{l_{mi}} - 1) \cdot l = T_i + \frac{\eta \cdot l}{l_{mi}} \cdot nSuccess - \eta \cdot l$

This leaves

$$\frac{\eta \cdot l}{l_{mi}} \cdot nSuccess = \theta \cdot nSuccess \tag{6.2}$$

as the only difficult part to implement in hardware. Assuming that l_{mi} is always a multiple of l and $\eta < 1$, the constant parameter θ is less than 1, and the term is a division. A division can be implemented very easy by shift operations if the divisor is a power of two. As the learning rate η is only a tuning parameter of the algorithm we can adjust it such that the constant is a power of two with only minimal influence on the algorithm.

The following example using the method proposed in Section 3.4.4 on page 56 will clarify the procedure. We choose an interval of length 80 ms and an accuracy of 0.1. Using Equation (3.13) the learning rate η is around 0.59. A standard CAN message with the maximal payload of 64 bit has a length of $64 + 47 = 111$ bits without bit stuffing. Assuming the pixel data is equally distributed, an average of 2 stuff bits will occur [NHNP01] resulting in an average message length of 113. With message length $l = 90400$ clock cycles ($113\,bit \cdot 800\,cycles/bit$) and an interval length l_{mi} of $8 \cdot 10^6$ clock cycles it results $\theta \approx \frac{1}{150}$. In our implementation, we used $\theta = \frac{1}{256}$ as the next lower power of two to ensure minimal hardware overhead while the required accuracy is met. Consequently, $\eta = 0.3456$ and $\eta \cdot l = 31250\,cycles$.

6.2.2 Experimental Results

In this section, we will show the behavior of the PPLA on our prototype. Instead of measuring the used bandwidth of each stream, we recorded the fps. The relation between both measures is linear, but the fps describe better the quality of the video application. The fps are measured by counting the number of messages that contain the beginning of a picture per second. The maximal fps that can be transmitted when using the full bandwidth and maximal CAN+ settings (details in Section 6.4) is 66 fps.

For the first experiment, we use the values derived in the previous section. By experiments we evaluated that $\epsilon = 0.1$ is the smallest value for our setup such that the error in measurement of the load has no influence on the results. The chosen accuracy $\mu = 0.1$ corresponds to an error of ± 6.6 fps. For the following experiments, we initially set all streams to transmit with 100 fps. This results in a full usage of the available bandwidth for the stream with the highest priority. At the start of the measurement, we enable the PPLA. Figure 6.4 shows the fps over time for two streams. The experiment shows that within 3 seconds the two streams share the available bandwidth fair. Figure 6.5 shows the same experiment for four streams, where the observations confirm our simulated and theoretical results: The bandwidth is shared fair with an

accuracy $\mu < 0.1 = 6.6\,fps$ and the convergence time increases linear with the number of streams.

As a second example, an accuracy of 0.01 should be kept. With a monitoring interval of $l_{mi} = 80\,ms$ and $l = 904\,\mu s$, according to Equation (3.11), the minimal error is $\frac{904\,\mu s}{80\,ms} = 0,0113$. Therefore, we increased the monitoring interval to $l_{mi} = 200\,ms$. Additionally, the same calculations for the parameters as for the previous experiment need to be performed:

- Equation (3.13): $\eta = 0.067$
- Equation (6.2): $\theta \approx \frac{1}{33268} \approx \frac{1}{32768} = \frac{1}{2^{15}}$
- $\eta * l = 6057$

In Figure 6.6 it is shown that the prototype can approve our desired property of an accuracy of 0.01 with the trade-off of longer convergence time.

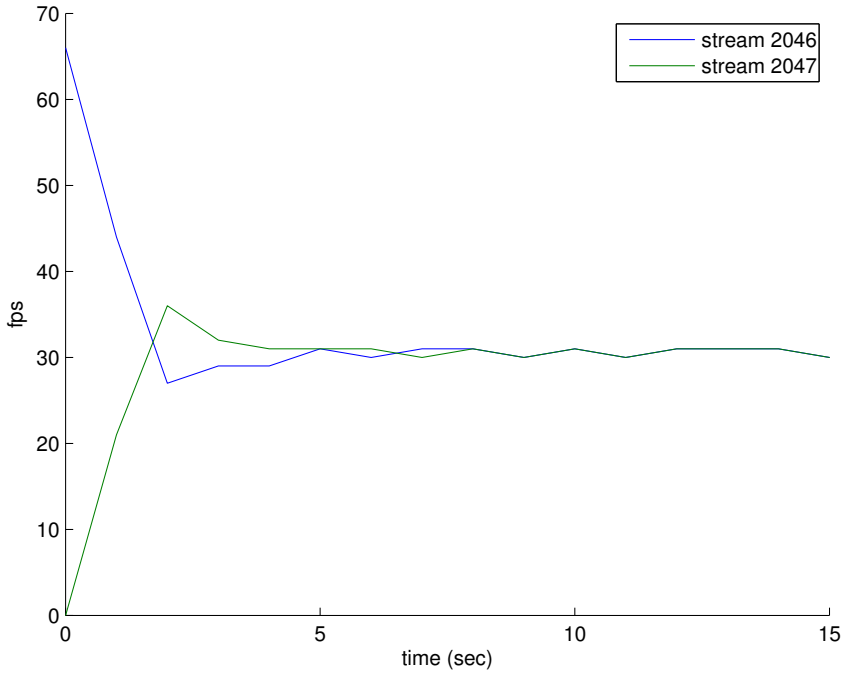


Figure 6.4: Fps as a function of time for two streams with the CAN id 2046 and 2047 when using the PPLA for $\mu = 0.1$.

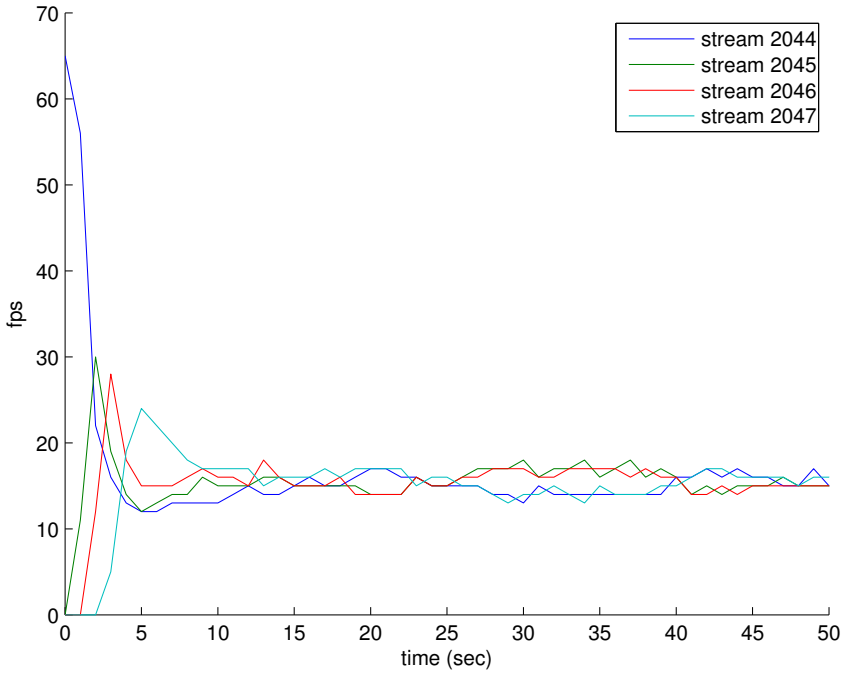


Figure 6.5: Fps as a function of time for four streams when using the PPLA for $\mu = 0.1$.

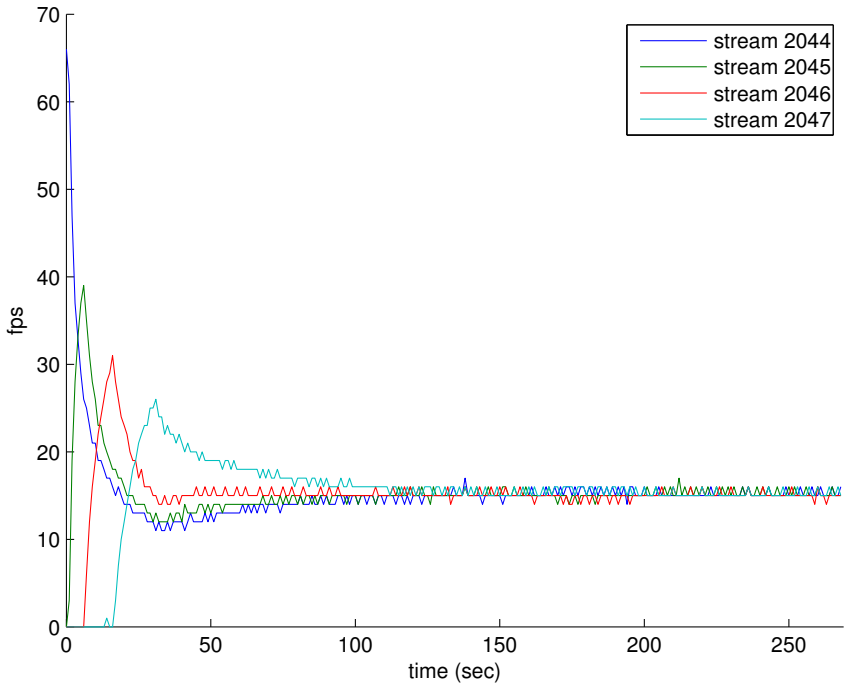


Figure 6.6: Fps as a function of time for four streams when using the PPLA for $\mu = 0.01$.

6.3 Dynamic Adaptation of Offsets

Chapter 4 shows that response times increase with higher bus utilization and dynamic adaptation of offsets can reduce this problem. Similar to other works [ZLH08, GSFR03, SREP08], we apply software simulation to evaluate the method. To compare the simulated to the real behavior, we use and extend the prototype system. Using the parallelism of the FPGAs and the CAN communication structure, the response times of the messages are measured and recorded for later evaluation without influencing the actual behavior of the system. Using this testbed with real components helps to reveal problems that are not directly visible in the simulation. We were able to identify that the clock drift of the individual nodes that is present in real systems has a strong influence on the response times. Furthermore, the worst case assumptions of the simulations lead in some cases to overestimation of the response times of up to factor two. Another advantage of a testbed is that it could be connected to a working CAN to analyze the response times of certain components when mixed with real communication traffic. It has to be noted that to the best of our knowledge this is the first physical setup of CAN where response times are measured.

In order to apply DynOAA in real distributed setting it is necessary to verify that the actual implementations with limited resources are able to perform similar to the simulation results. In addition, this has to be achieved within acceptable cost. As the acceptable solutions may be combinations of hardware and software, FPGAs prototyping is an almost ideal way to explore different solutions. The proposed DynOAA in Section 4.3.1 is not directly applicable to an embedded platform because of the too high memory and computation requirements. Therefore, we introduce and analyze purely software-based and hardware-based implementations of DynOAA. Then, based on these observations a mixed HW/SW solution is proposed. Finally, the different implementations are compared and tested by experiments.

6.3.1 Hardware Response Times Measurement

The testbed to measure the response times is based on the prototype system described in Section 6.1. However, the proposed testbed is designed to cope with any number of FPGA boards, only limited by the CAN protocol to 2032. The physical setup is only extended by a standard PC that is connected via RS232 to the DE2-115 board. The PC is used to gather and store the performance measurements. Each FPGA is representing one CAN node. Even though the area requirement for each node allows to implement several nodes on one FPGA, test results show that the performance results are the same to scheduling the identical streams on one node. This is due to the same used clock, which keeps the nodes on the same FPGA synchronized. In future work, it could be possible

to emulate several nodes in one FPGA by using different independent clock sources.

In the following, we first describe the design of the FPGA modules for the pure hardware implementation in detail. Then, the soft core based implementation is highlighted. As shown in Figure 6.7, a modular approach with simple interfaces is chosen to allow the easy replacement of any modules to research different behaviors in the future. The FPGA design is logical divided into three components: message generation, message transmission, and performance evaluation. The first two components represent the system under test, while the purpose of the last component is to measure the performance of the system under test.

6.3.1.1 Message Generation

The *message generation* represents the heart of the system under test. In a real system, the main functionality of the node is implemented here. In our testbed,

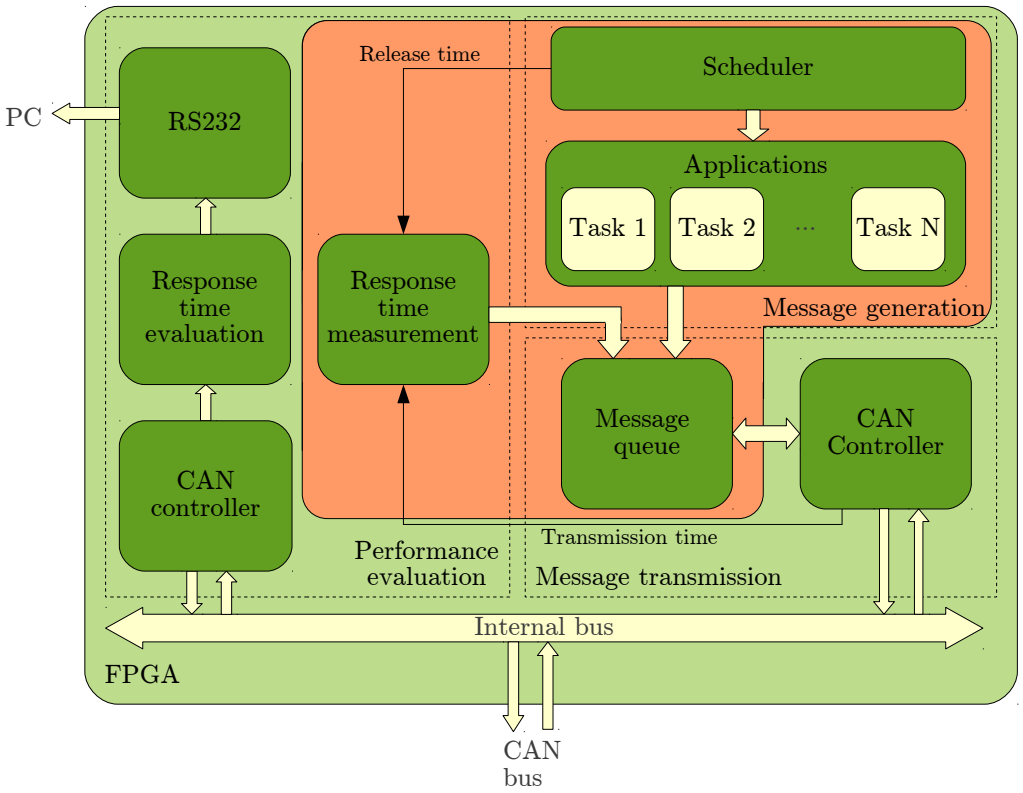


Figure 6.7: Schematic of the hardware modules used in each testbed FPGA.

it's function is to generate the communication traffic. This could vary from a simple pattern generator up to execution of real applications using dummy data.

In the current setup, a static cyclic scheduler is used that issues dummy tasks which do nothing but transmitting messages. These tasks fulfill the properties of the streams described in Section 2.1.2. According to the scenarios used for our tests the hyper period of the streams and therefore the period of our schedule is two seconds. A counter with a resolution of 8μs was chosen to carry out the schedule, which corresponds to the transmission time of one CAN bit. The embedded memory of the FPGA is used to store the information of the streams (identifier, next release time, period, length of the CAN message). Every counter tic, the stream memory is checked for a message to release. In case of a hit, the message is forwarded to the message transmission and the next release time is updated.

6.3.1.2 Message transmission

Two modules provide the functionality of the message transmission: The *message queue* and the *CAN controller*. The message queue stores the messages that are released and are waiting for transmission. Here, again different queuing schemes could be tested. We chose a priority queue for our case study as this represents the arbitration on the bus that would occur if each message stream would have its own CAN controller. The implemented message queue stores the waiting messages in the embedded memory. The priority schedule is enforced by scanning the whole memory with a maximum of 2032 entries every 8μs and picking the message with the highest priority. The priority is fixed and equal to the CAN identifier.

The actual transmission of the message is initiated by feeding the message information (identifier, message length, and data) to the CAN controller and enabling transmission. The CAN controller signals to the message queue whether it is ready for new transmissions and if messages were transmitted successfully, i.e., the arbitration was won. In this case, the transmitted messages can then be removed from the memory of the message queue. The function of the CAN controller module is to implement the CAN protocol. In our setup, we used our own developed CAN controller because it simplifies the response time measurement described in the following section. However, any CAN controller could be used. This could be a dedicated IC or an FPGA module like the CAN Protocol Controller from OpenCores [Moh].

6.3.1.3 Performance Evaluation

The performance evaluation component can be divided into one part for measuring of the response time, and one part for evaluation and transmission. The

response time measurement module is closely connected to the unit under test. The release time of the message is signaled by the scheduler. This release time is stored in an embedded memory of the FPGA. The starting time of the message transmission needs to be known to calculate the response time. Because of the arbitration mechanism of the CAN protocol, which cancels the transmission of messages with lower priority, the starting time can not be directly measured. Therefore, it is gathered by subtracting the transmission duration in bits, which is provided by our CAN controller, from the time of finishing the transmission. If the CAN controller would not provide the transmission duration, the start of transmission could be gathered by storing the start of transmission temporarily and validating it if the arbitration was successful.

The response times of all nodes are communicated by piggybacking the CAN messages. As soon as a response time calculation is finished the first four bytes of the next CAN message are used to transmit the response time. The 32 bits contain the corresponding identifier (11 bits) and the response time in μs with a maximum of two seconds (21 bits). So, the response time can be sent in all messages with a minimum length of four bytes. CAN messages that are smaller are extended to four bytes. The used scenarios and therefore typical automotive scenarios contain only very few messages ($<$20%) of smaller size, so this restriction of our approach is acceptable because it has a negligible influence on the measurements.

Independent FPGA modules are used to evaluate the response times and transmit the results for offline evaluation. This part of the performance evaluation needs to be implemented only once. An additional CAN controller is used to receive the response times and transmit it to the *response time evaluation* module. We used the AWW function from Equation (4.1) to evaluate the response time. The maximum response time for each stream is stored in an embedded memory. Every hyper period P, i.e., two seconds, the AWW function is calculated and sent out by using the RS232 module to transmit it to a PC. Then, the response time memory is cleared to be ready for the next measurements. The PC stores the values in a file for visualization or later evaluation.

6.3.1.4 Software Response Times Generation

On the DE2-115 board, we use a *Nios 2 processor* instantiated as so called soft core on the FPGA. To avoid too many hardware/software interfaces, we integrated the message generation, parts of the performance evaluation and parts of the message transmission into the processor as highlighted by the orange box in Figure 6.7. The CAN controller is not running in software, because the CAN protocol requires strict timing behavior which could not be kept by the used Nios 2 processor. There was no reason to implement the independent part of the performance evaluation also in software.

The functional behavior of the software-based CAN node is equal to the hardware-based one. However, as the message scheduler is run sequential on the processor and the interface to the CAN controller is interrupt driven, the timing behavior is not deterministic.

6.3.2 Embedded Systems Specifics and Requirements

DynOAA is not suitable for a real embedded implementation, because of the large amount of data needed to store the whole busy_idle_list in a monitoring interval. For example, for a monitoring interval of 2 seconds with a bit rate of 500 kbit/s there would be already $1 \cdot 10^6$elements in the list. A solution to this problem is to calculate $\alpha_{longest}$ and $\beta_{longest}$ continuously during the monitoring interval, which is shown in detail at the concrete implementation examples in Section 6.3.3.1 and 6.3.3.2.

For optimal performance the algorithm presented in Section 4.3.1 uses a circular busy_idle_list. In the practical implementation, the circular list only offers a minor performance improvement with a considerable implementation and computation effort. Therefore, in the following, a simple busy_idle_list is used where busy/idle times always begin and end at the beginning and the end of the monitoring interval.

Another issue is that in our model we assume it is possible to control the offsets of message streams. In a real application, the messages are triggered by tasks running periodically on the ECUs. Therefore, in order to control the message release and also the offset, we suggest doing it via controlling the task release. If a task has constant execution time, its release time can be simply calculated by subtracting the execution time from the desired message release time. If the task's execution time varies, the message release can be delayed by the worst case execution time, which obviously has to be available.

6.3.3 Design Alternatives for DynOAA

Figure 6.8 shows examples of three different implementation alternatives of ECUs. An ECU is typically a printed circuit board with integrated circuits (ICs) and other components. Besides power supply, sensors, and actuators the main functionality is implemented through the computations and communication of application tasks. Since we want to investigate the feasibility and performance of DynOAA, we will focus on these two aspects.

ECU 1 represents the currently most often used architecture, where one or several microcontrollers are responsible for the execution of the application tasks. Because of the limited computation power, the operation of the communication protocol needs dedicated hardware. This hardware unit is integrated into the microcontroller as indicated by the dashed box or is a separate IC. Finally,

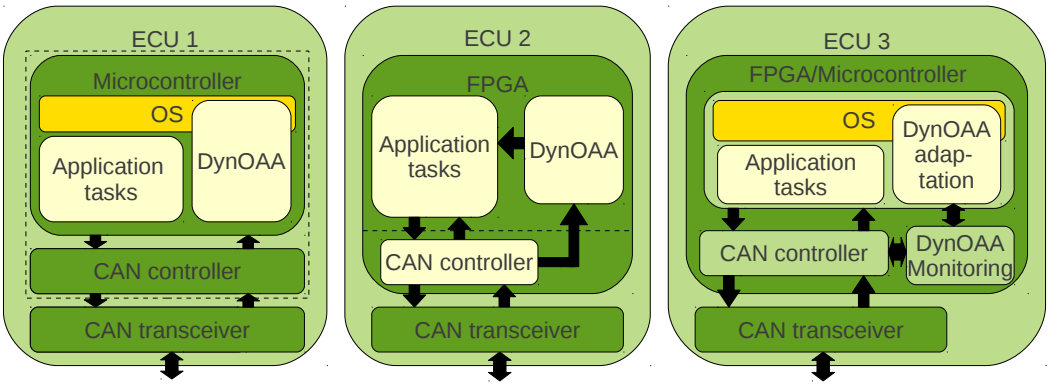


Figure 6.8: Different ECU architectures and how DynOAA could be integrated into them. The dashed boxes indicate a possible integration on a single IC.

a CAN transceiver [Inf08] is needed to establish the physical communication with the bus. In this setup type, typically an operating system is responsible for scheduling the application tasks. To include DynOAA, the releases of the tasks need to be influenced. This could either be done by integrating DynOAA into the operating system (as additional component of a scheduler) or by running DynOAA as an application task reporting the desired release times to the operating system.

ECU 2 uses an FPGA for computation and communication [ARC+07]. Here, the application tasks are implemented as hardware modules running in parallel to each other. Typically the application tasks are triggered periodically. In this type of ECU, DynOAA could be integrated by implementing it as a hardware component which triggers the application tasks with the adapted schedule. In this setup, the CAN controller could be implemented as dedicated hardware module on the FPGA similar to the hardware implemented application tasks.

ECU 3 represents a solution where DynOAA is implemented as a HW/SW solution. A general purpose DynOAA monitoring unit unloads the CPU of the microcontroller, while the adaptation is implemented as software integrated into the operating system. Details on this solution are provided in Section 6.3.3.3.

In the following sections, we describe in more details two implementation alternatives: (1) an interrupt-based implementation, which is better suited as software implementation on existing processors, and (2) a polling-based implementation, which is better suited as a hardware implementation that works in parallel with existing processors. In an interrupt-based implementation, DynOAA is executed when a specific event happens, thus resulting in reduced computation, but on the expense of more complicated control and potential degradation of performance of the remaining system. A polling-based implementation is bet-

ter suited for a hardware implementation, where the application and DynOAA module can run in parallel. Section 6.3.3.1 presents the interrupt-based implementation and Section 6.3.3.2 the polling-based implementation.

6.3.3.1 Software Implementation

Typical microcontrollers have processor cores ranging from simple 8-bit microprocessors to powerful 32-bit processors. In order to estimate the class of controllers DynOAA can be run on, we provide an interrupt-based alternative of DynOAA to measure the computational requirements. Interrupts are generated by the CAN controller and by a timer of the microcontroller. The pseudo code in Algorithm 6.2 describes these interrupt service routines. The CAN controller raises interrupts when the reception of a message starts and when it finishes. For example, the CAN controller inside the AT90CAN32/64/128 microcontroller series [Atm08] offers these interrupts. A timer with the period equal to the length of the monitoring interval is used to trigger the adaptation and to provide the current position when a CAN controller interrupt is raised. Therefore, the timer needs to be set to increment by one whenever one time slot passes, i.e., after transmission time of one CAN bit. During the traffic monitoring phase of DynOAA, the CAN controller interrupts are used to calculate $\alpha_{longest}$ and $\beta_{longest}$. When the traffic monitoring phase is finished, the timer interrupt starts the delay phase. This implementation of DynOAA requires storing the longest and the current busy and idle time. The two idle time variables require two integers for position of start (*.pos*) and length of the time interval (*.length*) with a maximum value equal to the length of the monitoring interval. The two busy time variables contain the length of the time interval (*.length*) and a Boolean whether the stream that is running the algorithm is sender of the first message of that busy time (*.isSender*). The id of the first message of the busy time has to be stored, if the ECU runs more than one stream. In addition, the position of start of the current busy time ($\beta_{current}.pos$) needs to be stored.

The principle of operation of the interrupt service routines is as follows. The timer interrupt (line 1) delays the task if it is the first of the longest busy time and resets all variables for the next monitoring phase. The start of reception interrupt (line 9) updates the longest idle time $\alpha_{longest}$ if necessary. In addition, the position and whether it is sender or not of the current busy time is updated, if the received message starts a new busy time. This is the case if the distance between the last two messages, i.e., $\alpha_{current}$, is greater than a threshold (*MESSAGESPACING*). In the ideal case, when transmission of delayed messages starts immediately this threshold would be 0. But depending on the accuracy of the involved microcontrollers the threshold needs to be increased. The end

Algorithm 6.2 Interrupt-based DynOAA implementation

```
1: // timer interrupt: adaptation/delay
2: if (timer = monitoring interval length) then
3:    if (β_longest.isSender) then
4:       next_position = α_longest.pos + α_longest.length / 2
5:       adapt(next_position mod period)
6:    end if
7:    reset(α/β_current/longest)
8: end if
9: // CAN controller interrupt: start of reception
10: if (start of reception) then
11:    α_current.length = currentCount − α_current.pos
12:    if α_longest.length < α_current.length then
13:       α_longest = α_current
14:    end if
15:    if (α_current.length > MESSAGESPACING) then
16:       β_current.pos = currentCount
17:       β_current.isSender = isCurrentSender()
18:    end if
19: end if
20: // CAN controller interrupt: end of reception
21: if (end of reception) then
22:    β_current.length = currentCount − β_current.pos
23:    if β_longest.length < β_current.length then
24:       β_longest = β_current
25:    end if
26:    α_current.pos = currentCount
27: end if
```

of reception interrupt (line 20) updates the longest busy time if necessary and stores the current timer position for the next interrupt.

In summary, the interrupt-based implementation requires seven unsigned integers and two Boolean values (or two 16-bit values, if several streams are adapted) to be stored in the memory of the microcontroller. In our example scenarios, when using the time of the transmission of one CAN bit per time slot on a 500 kbit/s CAN with a monitoring interval of 2 sec, the integer needs to have the maximum value

$$integer_{max} = \frac{2\,sec}{2\,\mu s} = 10^6 < 2^{20}. \tag{6.3}$$

The amount of computation can be predicted by assuming the worst case of sending the shortest message all the time. This means the CAN controller interrupts are raised with the period of the transmission time of the shortest message which is in our example around $80\,bits \cdot (\frac{1}{500\,kbit/s}) = 160\,\mu s$. During each interrupt only a few comparisons and subtractions need to be calculated. The timer interrupt is only raised every 2 seconds, so its computation requirements can be neglected.

In order to verify the amount of computation, we implemented the algorithm 6.2 on a Nios 2 32-bit processor instantiated on an Altera Cyclone II 2C70 FPGA running at 100 MHz. The used CAN controller is a VHDL module that is implemented on the FPGA as hardware. The memory requirement is minimal using seven 32-bit and two 16-bit integer that makes a total of 32 bytes. The number of cycles required to calculate the CAN controller interrupts (start (line 9) and end of reception (line 20)) is always below 2500 cycles ($25\,\mu s$). However, the overhead for entering and leaving the interrupt is in the order of $100\,\mu s$ that it dominates the calculation overhead. The timer interrupt that does the adaptation only needs around 1400 cycles ($14\,\mu s$) to calculate the required delay. Our example implementation uses a linked list of all streams ordered by the next occurrence to schedule the messages. The adaptation requires around 10,000 cycles to adapt the next occurrence of a stream. This means that the manipulation of the schedule dominates the computation overhead.

6.3.3.2 Hardware Implementation

A polling-based approach is perfectly suited for hardware implementation (see ECU 2 in Figure 6.8), because it does not use any processor resources and can be done in parallel with processor operation. It is therefore a good alternative to the interrupt-based approach. We prototyped this implementation in an FPGA and as such it can be used on ECUs that use FPGAs. However, the design could be easily ported on an ASIC, too. Algorithm 6.3 shows the pseudo code of this approach.

Algorithm 6.3 Polling-based DynOAA implementation

```
1: // monitoring:
2: init(α/β_current/longest)
3: for (each time slot) do
4:    update(α/β_current)
5:    if α_longest.length < α_current.length then
6:       α_longest = α_current
7:    end if
8:    if β_longest.length < β_current.length then
9:       β_longest = β_current
10:   end if
11: end for
12: // adaptation/delay:
13: if (β_longest.isSender) then
14:   next_position = α_longest.pos + α_longest.length / 2
15:   adapt(next_position mod period)
16: end if
```

In order to perform the algorithm, only the longest and the current busy and idle time ($\beta/\alpha_{current/longest}$) need to be stored similar to the software implementation in the previous section. At the beginning of each monitoring interval, all variables are initialized with a value of zero or false. During the monitoring phase, each time slot, $\alpha_{current}$ and $\beta_{current}$ are updated by simply counting up the length of $\alpha_{current}$ and reset $\beta_{current}$ when the current slot is idle. When the current slot is busy $\alpha_{current}$ is reset and the length of $\beta_{current}$ is incremented by one. In addition, $\alpha_{current}$ needs to update the position when a new idle time starts and $\beta_{current}$ needs to update *isSender* when a new busy time starts. After updating, the length of the current idle time $\alpha_{current}.length$ is compared with the length of the longest idle time $\alpha_{longest}.length$ and, if necessary, updated. The same is done for the busy times. During the adaptation, if the node is the sender of the first busy slot of the longest busy time, next_position is calculated and the node adapts by delaying its next release time according to formula 4.2.

In summary, this implementation of DynOAA requires six unsigned integers with a length of 20 bit according to Equation (6.3) and two Boolean values to be stored in registers. Similar to the software implementation, DynOAA can be used for several streams by storing the id of the first message of the busy time instead of only a Boolean. The amount of computation is very low, because in every time slot the FPGA carries out a few simple comparisons and increments,

and at the end of every monitoring interval carries out two integer additions, a division by two, and a modulo operation.

We designed the polling-based approach as hardware module in VHDL and implemented it on FPGA to verify the overhead. The algorithm is designed as a finite state machine using a counter, three flag bits, and 16 registers resulting in a memory requirement of 38 byte. Considering today's size and computation power of FPGAs the requirements for DynOAA in hardware can be neglected.

6.3.3.3 Hardware/Software-Solution

Running DynOAA as a purely software-based solution allows its straightforward integration to the currently typical ECUs. However, the analysis of the computation overhead shows, that the *monitoring* requires the most processing time as it is in the worst case triggered every 160 μs. It is necessary to raise an interrupt, because otherwise the monitoring information would be corrupted. The interrupt can, depending on the used microcontroller, cause a significant overhead. In addition, the use of interrupts can influence the timing behavior of the applications running on the microcontroller. Therefore, this part of DynOAA is only suited for powerful software processors or dedicated hardware.

The duration of the *delay* phase is mainly affected by the adaptation of the schedule. It is not necessary to trigger the adaptation by an interrupt, as it is not timing critical. Instead, we suggest integrating the delay phase into the scheduler of the time-triggered operating system.

The pure hardware solution is the most efficient one, as the required resources are negligible. If an FPGA with only hardware modules is used on the ECU this is the best solution. However, FPGAs are currently only used in very few embedded applications due to the different not established design method. An implementation of the ECU using an ASIC is also excluded because of the relatively low production volumes in the target domain of distributed control systems.

Due to these facts, we propose to implement the monitoring phase in a hardware unit and the delay phase in software as part of the operating system together with the applications (see ECU 3 in Figure 6.8). This design could be implemented on an FPGA using a softcore processor, but in the long run the hardware unit could be a general purpose hardware unit within a microcontroller. The hardware unit implements the polling-based approach in algorithm 6.3, and generates $\alpha_{longest}$ and the id of the first message of $\beta_{longest}$. A reset signal on the hardware unit resets $\alpha_{longest}$ and $\beta_{longest}$, and starts the next monitoring interval. The adaptation is triggered periodically by the operating system. It reads the values from the hardware unit, delays the appropriate stream and resets the hardware unit. This way the computation burden by DynOAA is reduced to almost zero by moving the computational intensive part to hardware. The

area requirement of the hardware unit is also very small. In order to be general purpose the hardware unit needs to be able to adjust the length of one time slot, i.e., length of the transmission of one CAN bit. Microcontrollers that include a CAN controller already have a register to set the speed of the CAN, which is read by the hardware unit.

6.3.4 Experimental Results

In the following, we will evaluate the response times measurement on the prototype system by means of the AWW according to Equation (4.1) and scenarios as described in Section 4.4.1. The results of a test setup with four FPGA nodes according to Figure 6.7 are compared to two different simulation engines: Our own, as described in Section 3.5.1, and *RTaW-sim* [RS09]. RTaW-sim is a freely available discrete-event simulation of the CAN bus, providing distributions and statistics of the average and maximum message response time from the start of the simulation. Due to these limited logging capabilities, we could only measure the AWW of the first hyper period and not use the advanced clock-drift simulations. The simulation model of both simulations are based on worst case assumptions of the CAN protocol. For example, the transmission length is calculated by assuming worst case bit-stuffing (all bits are equal).

Figure 6.9 and 6.10 show the results of measuring the AWW over time for a scenario with a bus load of 60% and 90%, *ls_60* and *ls_90* respectively. The offsets are chosen uniformly distributed as provided by Netcarbench [BHN07]. The measured results on the testbed are consistent with the lower values obtained through the simulations. Interestingly, the simulation results deviate up to a factor of two. This shows again how important physical setups are. For all runs, the simulations tend to be more pessimistic. One of the reasons could be that the simulations assume worst case bit-stuffing, so the transmissions length is always larger than the real one.

Another interesting observation is that with a bus load of 60%, there are larger variations than with a 90% bus load. As the testbed was designed to have a constant time from release to the first try of transmission on the bus, we could assure that the variations are caused by the traffic patterns on the bus.

The results also show, that the clock drift between the nodes has large influence on the AWW. We simulated this behavior in our own simulator by measuring the clock drift between the FPGA boards and change the periods accordingly. However, using fixed random offsets doesn't result in the same behavior, which can particularly be seen on the measurements with 90% bus load. It has to be noted that we only investigated on the clock drift after gathering the results on the testbed.

Next, results for a hardware-based and software-based DynOAA using an example implementation are shown. In order to compare both implementations

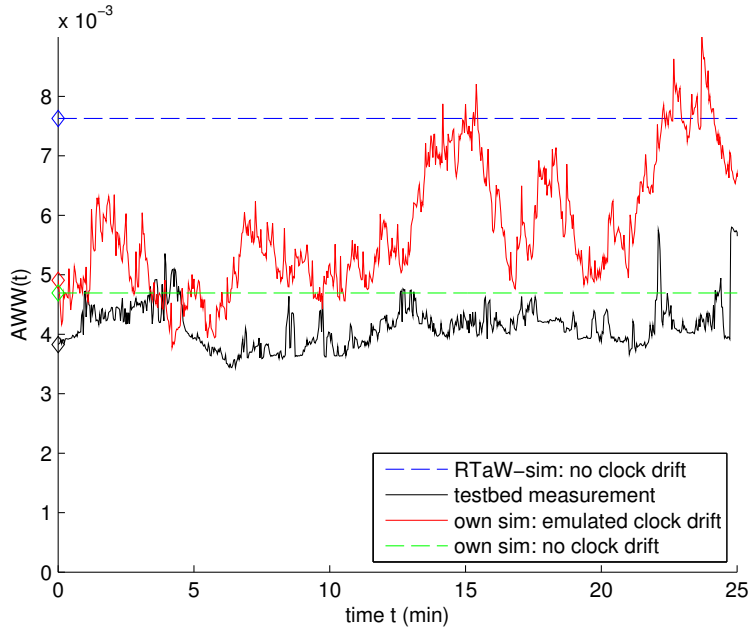


Figure 6.9: AWW over time for scenario ls_60 using random offsets.

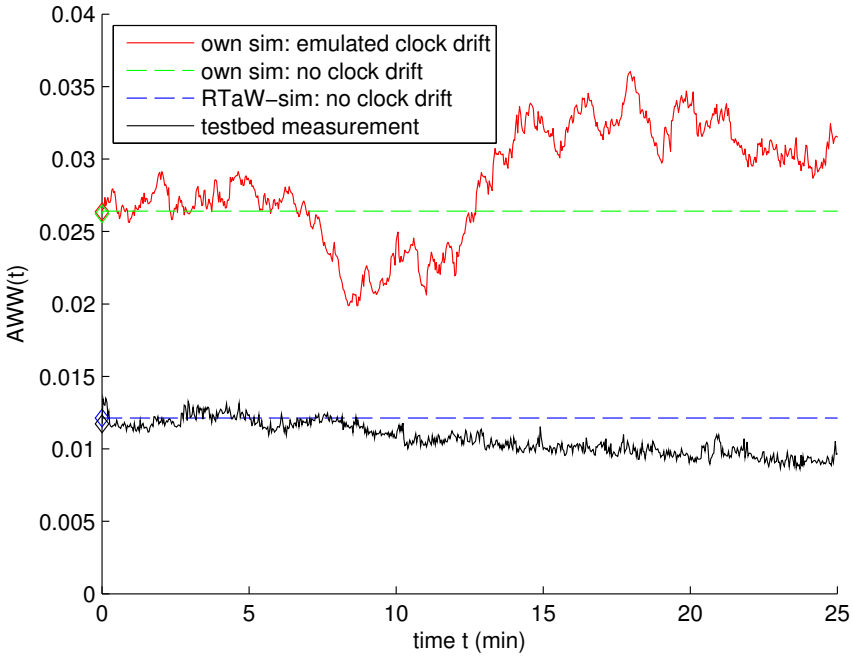


Figure 6.10: AWW over time for scenario ls_90 using random offsets.

we use a setup of two CAN nodes that communicate each to the other over a real CAN bus to explore the effects of a distributed embedded system as depicted in Figure 6.11. For the hardware-based implementation, we use two Xilinx XC5VLX100T FPGAs and the software-based implementation is run on a Nios 2 processor instantiated on two Altera Cyclone II 2C70 FPGAs.

Using two FPGAs, we can emulate close to real world examples as described in Section 4.4.1 from the automotive domain. In this section, we use scenario ls_60 as representative example. Figure 6.11 shows the setup of the prototype, where each ECU is implemented according to ECU 1 or 2 template in Figure 6.8. The streams are equally distributed on both ECUs. Each ECU module is implemented as described in Section 6.3.1 with the addition that the scheduler can use fixed offsets or the DynOAA as described in Section 6.3.3.

Using this setup, measurements were taken to compare the performance of DynOAA to fixed offsets using a hardware-based or software-based implementation as shown in Figure 6.12. The initial or fixed offsets are chosen uniformly distributed and are equal for all shown runs. The measurements show that using DynOAA improves the AWW in average almost by a factor of two. In addition, the behavior of using the hardware-based or software-based implementation is similar when using DynOAA. The difference between the hardware-based and software-based implementation when using fixed offsets is of pure random behavior, because the AWW is mainly influenced by the offset between both FPGAs. This offset is initially random depending on the sequence of turning on the FPGAs and then changes over time depending on the clock drift between the FPGAs. Nevertheless, the experiment shows that the hardware-based and software-based implementation produce almost equal results. Therefore, we analyze only the hardware-based implementation in the following.

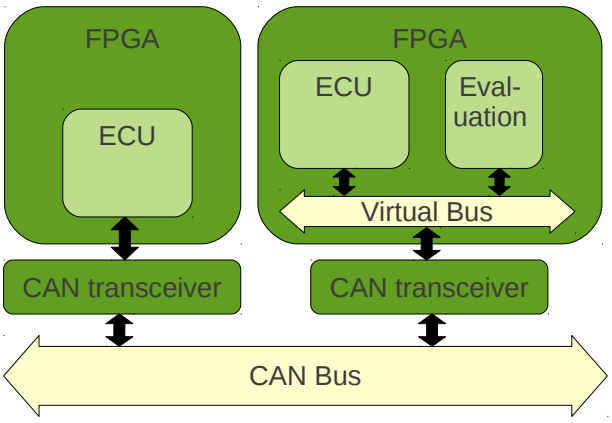


Figure 6.11: Schematic of the setup to compare the hardware-based and software-based implementation.

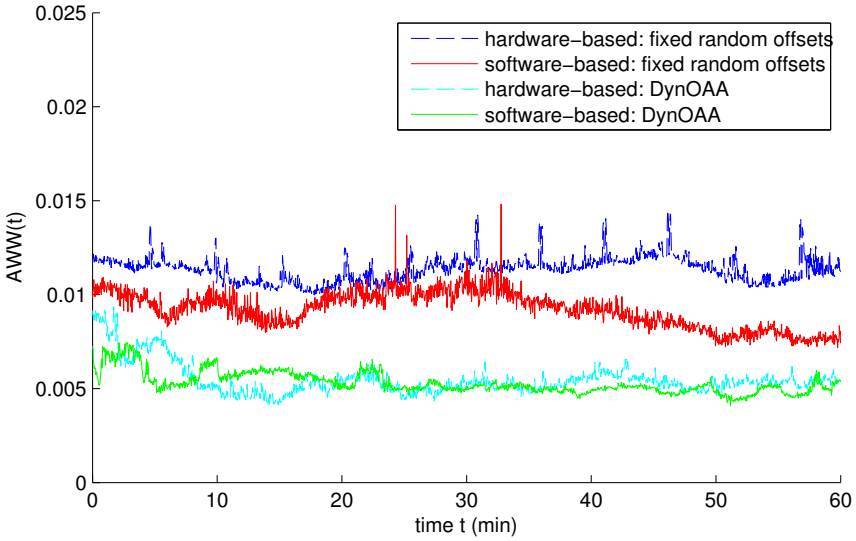


Figure 6.12: Value of the AWW over time for scenario hs_90 using DynOAA or fixed random offsets run as the hardware-based or software-based implementation.

Using four FPGAs, we ran the same scenarios as used in Figure 6.9 and Figure 6.10 with DynOAA enabled. The results in Figure 6.13 and Figure 6.14 show the reduction of the AWW compared to the initial random offsets.

Using two FPGAs, we demonstrate in the following experiment the interaction between DynOAA and the PPLA. Initially, the nodes generate traffic according

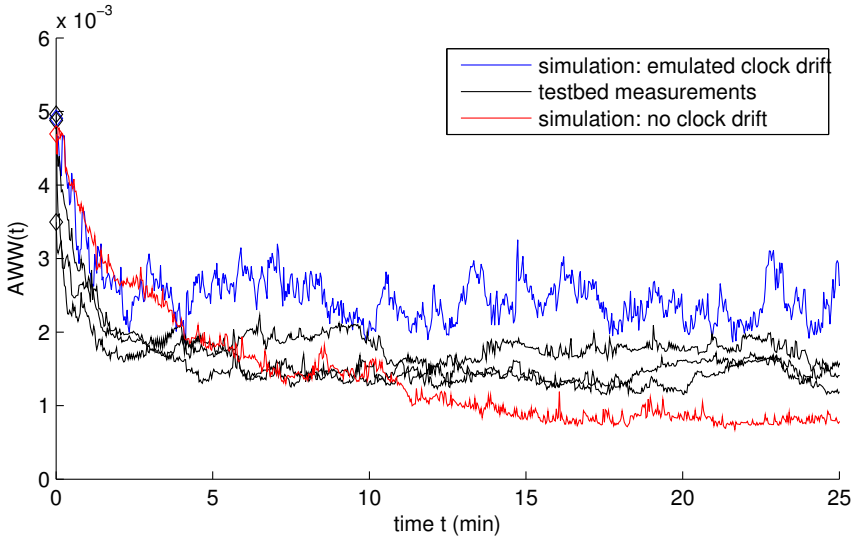


Figure 6.13: AWW over time for scenario ls_60 using DynOAA.

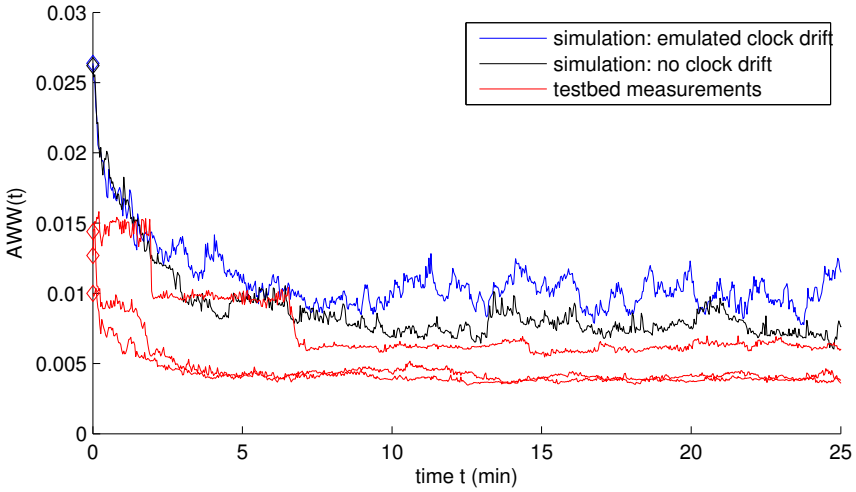


Figure 6.14: AWW over time for scenario ls_90 using DynOAA.

to scenario ls_60 where the AWW is improved by DynOAA. After 55 seconds two video streams are added using the PPLA to adjust the sending period. After 93 seconds the traffic of scenario ls_60 is stopped and the two video streams share the full bandwidth. Figure 6.15 shows the fps and the AWW over time. The experiment shows that the video streams share the bandwidth fair even under changing traffic with higher priority. Furthermore, it has to be noted that the video streams that constantly send messages increase the AWW such that the positive effect of DynOAA is neutralized. However, this increase also occurs if random offsets are used. The reason for it is that the nonpreemptive arbitration mechanism causes that messages that are released during the transmission of lower priority messages to wait until they are finished the transmission.

6.4 CAN+

First results on using CAN+ are shown in Section 5.4.2. In this section, we provide further evidence of the performance of CAN+. As the prototype system is based on low priced FPGAs, we can show the low hardware requirements of CAN+. Additionally, if we would only use standard CAN it would not be possible to transmit videos with reasonable quality and a visualization of the fair bandwidth distribution would be difficult.

In contrast to the previous results, we use the standard CAN bit rate of 125 kbit/s instead of 1 Mbit/s. This turned out be an advantage, because the number of overclocked bits can be adjusted finer, so that the gray zone can be used better. The resulting data rate of CAN+ is the same independent of the

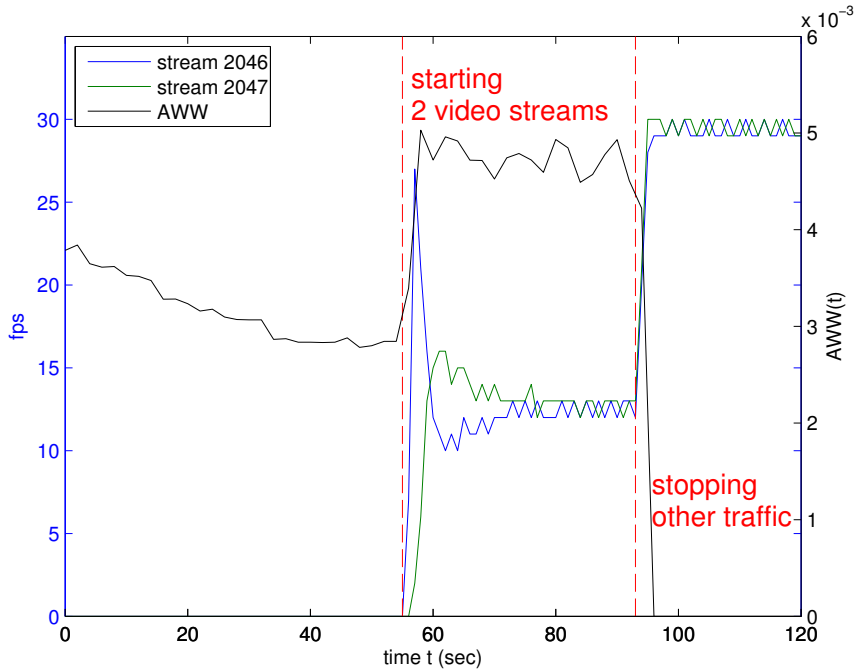


Figure 6.15: Fps and AWW over time for scenario ls_60 with DynOAA when adding two video streams using PPLA.

standard CAN bit rate, because the data rate of CAN+ depends only on the amount of time spend in the gray zone in relation to the time not spend in the gray zone. This ratio is the same for all standard CAN bit rates because the change in bit rate influences the time spend in the gray zone and the time not spend in the gray zone equally. A further difference to the previous results is the use of more than two boards, which shows the robustness of the protocol. The DE2-115 board only supports standard CAN and generates traffic typical for standard CAN buses that demonstrates again the backward-compatibility of the approach.

Similar to the measurements in Section 5.4.1, we test by experiments how the parameters of the CAN+ protocol are set to achieve a maximal data rate for our setup. In contrast to the previous measurements the focus lies on the maximal data rate assuming that the standard CAN nodes use the maximal compatible setup as explored in Section 5.4.1. The quality criteria in this setup is the percentage of corrupted overclocked bits. We measure this by transmitting CAN+ messages with a predefined content, such that the receiver knows the content and can check whether it is transmitted correctly or not. An eight bit value counted up is used as predefined value. The error counter is read to the PC by an integrated *chipscope [chi]* core using the *JTAG* [jta01] connection. The error count is incremented each time one eight bit value is not correct.

The following four parameter need to be chosen: Start of the gray zone t_s, end of the gray zone t_e, length of one overclocked bit $\hat{t}_{bit}$ and overclocking factor ζ. t_e depends on the other three parameter: $t_e = t_s + (\zeta + 1) \cdot \hat{t}_{bit}$. The module Bit Handler (cf. Figure 5.5) operates at a frequency of 200 MHz. Therefore, t_s and $\hat{t}_{bit}$ have the granularity of 5 ns and are in the following stated in clock cycles. The first run of experiments shows the bit error rate depending on the length of one overclocked bit $\hat{t}_{bit}$. Start of the gray zone t_s is set to 300 cycles and overclocking factor ζ to 50 to eliminate errors due to exceeding the gray zone. We transmitted 10 mins with 416 kbyte per second that is a total of around 250 Mbyte. This corresponds to maximal bus load. The results are listed in Table 6.1. If the application, such as our example video transmission, tolerates a small percentage of errors, $\hat{t}_{bit} = 4\,cycles = 20\,ns$ is the best option.

The size of the gray zone, i.e., start of the gray zone t_s and overclocking factor ζ, is only limited by the standard CAN nodes included in the network. In the prototype platform, the standard CAN node is the CANcaseXL as already used and evaluated in Section 5.4.1. For the CAN controller setting of the CANcaseXL with the time of the sampling point at 87% of the bit time we achieved the best results. We tested the size of the gray zone with the setting $\zeta = 100\,cycles$ and $\hat{t}_{bit} = 4\,cycles$. Then, this interval of overclocked bits was moved as close as possible without errors to the beginning and to the end of the transmission interval of one CAN bit. It was possible to set $t_s = 150\,cycles$ and $t_e = 1100\,cycles$. If $\hat{t}_{bit} = 4\,cycles$ this would result in an overclocking factor of $\zeta = 237$. However, the FPGA synthesis could not solve the timing constraints for $\zeta > 180$. So, the best result we could achieve with the prototype was $\zeta = 180$, which is a usable data rate of around 1.4 Mbyte/s.

The quality of the wiring is important and can increase the error rate or reduce the size of the gray zone if it doesn't meet the CAN specification. Nevertheless, the previous tests were conducted on a simple unshielded twisted pair cable soldered by hand. We also tested a flat wire cable which showed almost similar results. It also has to be noted that these measurements are only valid for this particular setup and should only present an estimate of the possible performance.

$\hat{t}_{bit}$	2	3	4	5	6
Absolute number of erroneous bytes	$1.34 \cdot 10^8$	$2.29 \cdot 10^7$	4	0	0
Error in percentage	51.20	8.76	$1.52 \cdot 10^{-6}$	0	0

Table 6.1: Erroneous bytes for CAN+ transmission for different $\hat{t}_{bit}$.

7 Conclusions

This thesis presents methods to improve the communication in Distributed Embedded Systems (DES). DES targeted in this thesis are embedded systems that are connected by a communication bus.

Design of DES is categorized into the design of static and dynamic systems. While static approaches try to consider all system states during design and, therefore, can guarantee certain behavior, they often are too pessimistic and don't make full use of the available resources. In contrast, dynamic approaches react to changes during run-time of the system. Currently, dynamic approaches are small building bricks in a static design flow. However, with the increasing size of DES it is necessary to further develop dynamic approaches up to a point where the system is able to self-organize.

The applications on DES are usually spread across multiple nodes. This makes the communication of the tasks of the application a critical factor. With the increase of applications and therefore increase of communication demand, it becomes a difficult task to fulfill the communication requirements. A model based on message streams is defined that helps to focus the work on the communication aspects. Soft real-time streams present applications that transmit messages periodically. For them it is important that the response time is minimized and kept within certain limits. In contrast, for bandwidth streams the amount of messages that can be transmitted (i.e. throughput) is significant.

A further concentration is done by looking only at bus-based protocols that support an event-triggered priority-based access mechanism. Three such protocols are presented that are able to use the methods developed in this thesis:

CAN, Widom, and Flexray. Here, in particular the CAN protocol is explained in detail, as it is used as running example throughout this thesis.

The problem with bandwidth streams is that priority-based protocols require different priorities for each stream. Therefore, fair bandwidth sharing is not directly possible. It is considered fair if each stream gets the same amount of bandwidth. On an existing game theoretical model for medium access, the contention-based medium access was replaced by priority-based access and the payoff was extended. Using this extended model, it could be proven that a fair distribution is a stable state. Based on this, a distributed algorithm is developed that converges to this stable state without requiring any additional communication. Two variants of the algorithm are proposed using either probabilistic or periodic access. While the probabilistic access algorithm allows proving of the fair bandwidth sharing properties, the periodic access algorithm performs better in terms of convergence and accuracy, and predictability by worst case analysis. Extensive simulations on the game theoretic model and the CAN protocol show the performance with different parameter settings.

The fixed static priorities on the priority-based access mechanisms have the advantage to guarantee deterministic response times on the one hand, because access is always given to the message with the highest priority. On the other hand, low priority messages can have huge response times and even can starve leading to a violation of soft real-time requirements. The worst case scenario, where each message has the worst case response time, is when all messages are released at the same time. However, this assumption is too strong for messages released on different nodes due to the inherent asynchronous nature of DES. Furthermore, for periodic message streams it is possible to distribute accesses to the bus in time to minimize the probability of simultaneous accesses and conflicts. The distribution of accesses is achieved by setting different offsets between the message streams. What makes the problem difficult is the asynchronous nature of DES caused by different clocks that drive the nodes and the lack of a global reference clock. Current approaches assign statically calculated offsets of messages that are assumed to be synchronous by using offline heuristics. Assigning offsets to the messages results in better response times, but does not take into account the asynchronous nature of the nodes. They also do not consider the dynamics of the system operation and resulting traffic. To overcome these disadvantages an algorithm to dynamically adapt the offsets of the message streams during run-time based on monitoring of the network traffic is proposed. The algorithm can be done by software routines within the current message format and with no modifications of the protocol or the hardware.

Software simulations and evaluation on a prototype show that this approach achieves reduction of the average message response times and enables the use of the same bus at higher bus loads. As a consequence, the bus can be used in applications in which new nodes will be added as the application develops with

the confidence that the system will adapt to the system configuration (number of streams) as it changes. Prototype implementations on different platforms show that the algorithm can be executed in software, but a hardware implementation could provide the same functionality with much less resources.

The increasing size and complexity of DES, and new applications demand for higher data rates. Besides using the available communication resources carefully on software level, modification on protocol level is a valid option. Therefore, an extension of the CAN protocol, called CAN+, is introduced that allows transmission with a higher data rate. The increase of data rate is achieved by transmitting additional information during unused time intervals. This allows that nodes that use the standard CAN protocol are not interfered and heterogeneous networks can be built. Thus, existing systems can be reused and hardware costs can be reduced for applications with low data rate demands. The protocol extension was implemented on several prototyping platforms, showing a speedup of up to 16 times the standard CAN data rate.

Even though CAN is widely used and well established in the industry, this thesis showed that there is still space for improvement. Especially the trend towards self-organization offers new possibilities because of the unmanageable complexity of current DES. The presented methods show that self-organization can be integrated almost for free and provides scalable and flexible systems.

7.1 Future Work

The main part of future work would be to test and extend the developed methods to other protocols than CAN. First, closely related protocols such as Flexray or Widom should be evaluated. While the almost identical arbitration mechanism on Widom should allow a simple porting, the token-based access of Flexray will require more research. Furthermore, the use of the here presented methods could be an option for priority-based on-chip buses.

Until now, the sharing of bandwidth only considers fair, i.e., equal, sharing of bandwidth. However, it could be imaginable that some applications require less bandwidth than others. For example a video stream requires more bandwidth than an audio stream for the same level of quality. Therefore, it would be desirable to be able to weight the applications. First results show that adding a simple weight multiplier to the adaption of the learning algorithms does not generate the desired behavior. So, further research is necessary.

The model in this thesis only considers the communication. This is certainly reasonable for showing the performance of the scheduling algorithms. Nevertheless, it would be interesting how the total response times behave, which also consists of the calculation time of the task on the node. Here, it would be particularly interesting how the relation between calculation and communication delay

is influenced. Furthermore, an analysis whether and how data dependencies of the distributed tasks effect the offset adaption could be carried out.

The introduced method of increasing the data rate requires an increase of bit transmission rate during certain time intervals. This is achieved by shortening the transmission time of one bit. The consequence is that the transmission is more susceptible to errors. Even the standard CAN protocol provides error detection codes. As future work, error detection or even correction codes could be in included. Furthermore, the trade-off between data rate and error rate needs to be evaluated. It is even imaginable to use a dynamically changing data rate depending on the physical conditions.

By implementing the proposed methods in a realistic prototype setup, a big step towards integrating self-organizing mechanisms in real DES is done. It would be desirable to see the next generation of DES using them.

German Part

Selbstorganisation und Optimierung von prioritätsbasierten Kommunikationsbussen

Zusammenfassung

In der vorliegenden Arbeit werden Methoden zur Verbesserung der Kommunikation in verteilten eingebetteten Systemen (DES) vorgestellt. Dazu werden zuerst die Zielsysteme eingeführt und definiert. Die hier untersuchten DES sind eingebettete Systeme, die durch einen Kommunikationsbus verbunden sind.

Der Entwurf von DES wird in statische und dynamische Ansätze unterteilt. Statische Ansätze versuchen alle möglichen Systemzustände zu berücksichtigen, weswegen es möglich ist, das Verhalten zu garantieren. Dies geschieht allerdings auf Kosten der Effizienz, da durch pessimistische Annahmen die zur Verfügung stehenden Ressourcen nicht voll ausgenutzt werden. Im Gegensatz dazu ermöglichen dynamische Ansätze auf Änderungen im System zur Laufzeit zu reagieren. Heutzutage stellen dynamische Ansätze nur kleine Bausteine in einem statischen Entwurfsfluss dar. Allerdings wird es mit der steigenden Anzahl an Komponenten und damit Komplexität in DES nötig sein, dynamische Ansätze weiter zu entwickeln. Dies könnte so weit führen, dass sich das System selbstorganisiert.

Anwendungen in DES sind meistens über mehrere Knoten verteilt. Das bedeutet, dass die Kommunikation zwischen den Tasks der Anwendung ein entscheidender Faktor ist. Mit Zunahme der Anzahl der Anwendungen steigt auch der Bedarf an Kommunikation. Dies erschwert die Erfüllung der Anforderungen. Ein Modell basierend auf Nachrichtenströmen wird definiert, so dass die Konzentration auf die Kommunikationsaspekte möglich ist. Nachrichtenströme mit weicher Echtzeitanforderung stellen Anwendungen dar, die periodisch Nachrichten versenden und die Antwortzeit dieser minimiert werden soll. Im Gegensatz dazu ist es für Nachrichtenströme mit Bandbreitenanforderung wichtig, dass eine gewisse Anzahl von Nachrichten in einer festgelegten Zeit übertragen wird.

Des Weiteren konzentriert sich die Arbeit auf Bus-basierte Protokolle, die ereignisgesteuerte Zugriffe durch Prioritäten zulässt. Drei derartige Protokolle werden vorgestellt: CAN, Widom und Flexray. Mit ihnen können die in dieser Arbeit entwickelten Methoden umgesetzt werden. Hierbei wird insbesondere das Protokoll CAN detailliert beschrieben, da es als immer wiederkehrendes Beispiel in dieser Arbeit verwendet wird.

Bedingt durch das prioritätsbasierte Protokoll muss man den Nachrichtenströmen unterschiedlichen Prioritäten zuteilen. Diese Tatsache stellt für Nachrichtenströmen mit Bandbreitenanforderung ein Problem dar, da dadurch eine

faire Bandbreitenaufteilung nicht möglich ist. Eine faire Bandbreitenaufteilung liegt vor, wenn alle Nachrichtenströme die gleiche Datenbandbreite bekommen.

Um dieses Problem bestmöglichst analysieren zu können, wurde auf ein bereits bestehendes spieltheoretisches Modell zurückgegriffen. Das konfliktbasierte Zugriffsverfahren wurde durch den prioritätsbasierten Zugriff ersetzt und in der Nutzenfunktion erweitert. Basierend auf diesem erweiterten Modell kann bewiesen werden, dass die faire Vertailung der Bandbreite einen stabilen Systemzustand darstellt. Infolge dieser Erkenntnis wird ein Algorithmus entwickelt der diesen stabilen Zustand ohne zusätzliche Kommunikation erreicht. Zwei Varianten des Algorithmus werden vorgestellt, welche entweder einen wahrscheinlichkeitsbedingten oder periodischen Zugriff verwenden. Während der wahrscheinlichkeitsbedingte Zugriff eine Beweisbarkeit der Eigenschaften ermöglicht, ist die Geschwindigkeit der Konvergenz und die Genauigkeit beim periodischen Zugriff besser. Außerdem ermöglicht der periodische Zugriff eine Vorhersagbarkeit durch analytische Berechnung des ungünstigsten Falles. Ausführliche Simulationen anhand des spieltheoretischen Modells und des CAN Protokolls zeigen die Unterschiede bei verschiedenen Parametereinstellungen.

Die festen statischen Prioritäten des prioritätsbasierten Zugriffsverfahrens haben einerseits den Vorteil, dass deterministische Antwortzeiten garantiert werden können, da den Zugriff immer die Nachricht mit der höchsten Priorität hat. Andererseits führt es aber dazu, dass Nachrichten mit niedrigen Prioritäten sehr lange Antwortzeiten haben können, was sogar dazu führen kann, dass weiche Echtzeitanforderungen verletzt werden. Der schlimmstmögliche Fall, bei dem die Antwortzeiten am längsten sind tritt ein, wenn alle Nachrichten gleichzeitig versendet werden. Diese Annahme ist allerdings unrealistisch für Systeme, bei denen die Nachrichten von verteilten asynchronen Knoten versendet werden. Außerdem, kann für periodische Nachrichtenströme der Zugriff so verteilt werden, dass die Wahrscheinlichkeit für gleichzeitige Zugriffe minimiert wird. Die Verteilung wird erreicht indem unterschiedliche Versätze zwischen den Nachrichtenströmen eingestellt werden. Die Schwierigkeit dabei ist, dass die Asynchronität der DES eine ständige Verschiebung der Versätze verursacht. Heutige Ansätze vergeben statisch berechnete Versätze für Teile des Systems die synchron angenommen werde können. Dadurch werden bessere Antwortzeiten erreicht, allerdings wird die asynchrone Natur der Knoten nicht einbezogen. Außerdem werden die dynamischen Änderungen des Systems zur Laufzeit und der daraus veränderte Verkehr nicht berücksichtigt. Um diese Nachteile zu vermeiden wird ein Algorithmus vorgeschlagen, der die Versätze der Nachrichtenströme während der Laufzeit auf Grund des aktuellen Busverkehrs anpasst. Dieser Algorithmus kann alleine durch Softwareroutinen mit dem bestehenden Nachrichtenformat durchgeführt werden, ohne dabei das Protokoll oder die Hardware zu modifizieren.

Softwaresimulation und Evaluierung auf einem Prototyp zeigen, dass dieser Ansatz eine Verkürzung der durchschnittlichen Antwortzeit erreicht. So wird ermöglicht, den gleichen Bus mit höherer Auslastung zu verwenden. Daraus resultiert, dass der Bus in Anwendungen zum Einsatz kommen kann, bei dem neue Knoten während der Entwicklung hinzugefügt werden, da das System auf die neue Konfiguration (Anzahl der Nachrichtenströme) reagiert. Prototypen auf unterschiedlichen Plattformen zeigen, dass der Algorithmus in Software ausgeführt werden kann. Eine Hardwareimplementierung bietet allerdings die gleiche Funktionalität mit weniger Ressourcen.

Die steigende Größe und Komplexität von DES durch ständig neue Anwendungen erhöht den Bedarf an höheren Datenraten. Neben der Möglichkeit, die vorhanden Kommunikationsressourcen mit Bedacht auf Softwareebene zu nutzen, ist die Modifikation auf Protokollebene eine weitere Lösung. Aus diesem Grund wird in dieser Arbeit eine Erweiterung des CAN Protokolls vorgestellt, mit der eine höhere Datenrate möglich ist: CAN+. Diese Erhöhung wird erreicht, indem zusätzliche Daten während ungenutzten Zeitintervallen übertragen werden. Damit ist es möglich Knoten, die das Standard CAN Protokoll nutzen in heterogenen Netzwerken weiterhin zu integrieren. Somit können existierende Systeme weiter genutzt werden und Kosten für Anwendungen, die lediglich niedrige Datenraten benötigen reduziert werden. Die Protokollerweiterung wurde auf mehreren Plattformen getestet und es konnte eine 16-fache Beschleunigung im Vergleich zu Standard CAN erreicht werden.

Bibliography

[Alb04] A. Albert. Comparison of event-triggered and time-triggered concepts with regard to distributed control systems. *Embedded World*, 2004:235–252, 2004.

[APF02] L. Almeida, P. Pedreiras, and J.A.G. Fonseca. The FTT-CAN protocol: why and how. *Industrial Electronics, IEEE Transactions on*, 49(6):1189 – 1201, December 2002.

[ARC+07] R. Anthony, A. Rettberg, D. Chen, I. Jahnich, G. de Boer, and C. Ekelin. Towards a dynamically reconfigurable automotive control system architecture. *Embedded System Design: Topics, Techniques and Trends*, pages 71–84, 2007.

[Ass91] Electronic Industries Association. Interface between data terminal equipment and data circuit-terminating equipment employing serial binary data interchange, revised from eia232d. 1991.

[ATDP10] H. Aysan, A. Thekkilakattil, R. Dobrin, and S. Punnekkat. Efficient fault tolerant scheduling on controller area network (can). In *Emerging Technologies and Factory Automation (ETFA)*, pages 1 –8, sept. 2010.

[Atm08] Atmel Corporation. *8-bit AVR Microcontroller with 32K/64K/128K Bytes of ISP Flash and CAN Controller*, 2008.

[Aud] Audi AG.

[Aud91] N.C. Audsley. *Optimal priority assignment and feasibility of static priority tasks with arbitrary start times.* Citeseer, 1991.

[AV10] Ian Akyildiz and Mehmet Can Vuran. *Wireless Sensor Networks.* John Wiley & Sons, Inc., New York, NY, USA, 2010.

[BBD06] L. Buşoniu, R. Babuška, and B. De Schutter. Multi-Agent Reinforcement Learning: A Survey. In *Proceedings of the 9th International Conference on Control, Automation, Robotics and Vision (ICARCV 2006)*, pages 527–532, Singapore, December 2006.

[BBDS08] L. Busoniu, R. Babuska, and B. De Schutter. A comprehensive survey of multiagent reinforcement learning. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 38(2):156 –172, march 2008.

[BGH92] D.P. Bertsekas, R.G. Gallager, and P. Humblet. *Data networks*, volume 2. Prentice-Hall International, 1992.

[BHN07] C. Braun, L. Havet, and N. Navet. NETCARBENCH: A benchmark for techniques and tools used in the design of automotive communication systems. In *7th IFAC International Conference on Fieldbuses and Networks in Industrial and Embedded Systems*, pages 321–328, 2007.

[BHR+09] E.J. Bueno, A. Hernandez, F.J. Rodriguez, C. Giron, R. Mateos, and S. Cobreces. A dsp- and fpga-based industrial control with high-speed communication interfaces for grid converters applied to distributed power generation systems. *Industrial Electronics, IEEE Transactions on*, 56(3):654 –669, 2009.

[BHWY05] T. Bai, L. Hu, Z. Wu, and G. Yang. Flexible fuzzy priority scheduling of the CAN bus. *Asian Journal of Control*, 7(4):401–413, 2005.

[BLCA02] G.C. Buttazzo, G. Lipari, M. Caccamo, and L. Abeni. Elastic scheduling for flexible workload management. *IEEE Transactions on Computers*, pages 289–302, 2002.

[CAN91] CAN specification 2.0 b. *Robert Bosch GmbH, Stuttgart, Germany*, 1991.

[CAN11] Can with flexible data-rate - white paper version 1.1. *Robert Bosch GmbH, Stuttgart, Germany*, 2011.

[Cat05] J. Catsoulis. *Designing embedded hardware*. O'Reilly Media, Inc., 2005.

[CCLD07] L. Chen, T. Cui, S. H. Low, and J. C. Doyle. A Game-Theoretic Model for Medium Access Control. In *Proceedings of International Wireless Internet Conference (WICON)*, Austin, TX, October 2007. (invited).

[Cha94] J. Charzinski. Performance of the error detection mechanisms in can. In *Proceedings of the 1st International CAN Conference*, volume 9, 1994.

[chi] Chipscope. `http://www.xilinx.com/tools/cspro.htm`.

[CKT03] S. Chakraborty, S. Künzli, and L. Thiele. A general framework for analysing system properties in platform-based embedded system designs. In *Proc. 6th Design, Automation and Test in Europe (DATE)*, pages 190–195. Citeseer, 2003.

[Cor08] Steve Corrigan. *Controller Area Network Physical Layer Requirements*. Texas Instruments Incorporated, 2008.

[Cro] Crossbow Technology, Inc. *MICAz - wireless measurement system*. `http://www.openautomation.net/uploadsproductos/micaz_datasheet.pdf`.

[CV95] G. Cena and A. Valenzano. A distributed mechanism to improve fairness in can networks. In *Factory Communication Systems, 1995. WFCS '95, Proceedings., 1995 IEEE International Workshop on*, pages 3 –11, October 1995.

[CV99] G. Cena and A. Valenzano. Overclocking of controller area networks. *Electronics Letters*, 35(22):1923–1925, 28 Oct 1999.

[CV02] G. Cena and A. Valenzano. Achieving round-robin access in controller area networks. *Industrial Electronics, IEEE Transactions on*, 49(6):1202 – 1213, dec 2002.

[DA10] H.A. Dharmawan and S.A.M. Ali. A modular approach and controller area network bus implementation in an embedded system of a 3-phase 10 kva energy efficient switchable distribution transformer. In *Industrial Electronics and Applications (ICIEA), 2010 the 5th IEEE Conference on*, pages 62 –67, june 2010.

[DBBL07] R.I. Davis, A. Burns, R.J. Bril, and J.J. Lukkien. Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised. *Real-Time Systems*, 35(3):239–272, 2007.

[DDNDM07] Davide Devescovi, Elisabetta Di Nitto, Daniel Dubois, and Raffaela Mirandola. Self-organization algorithms for autonomic systems in the selflet approach. In *Proceedings of the 1st international conference on Autonomic computing and communication*

systems, Autonomics '07, pages 26:1–26:10, ICST, Brussels, Belgium, Belgium, 2007. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[DFG11] Deutsche Forschungsgemeinschaft, DFG SPP 1183 - Organic Computing Initiative. `http://www.organic-computing.de/SPP`, 2011.

[DIN03] DIN EN 55025: Funk-Entstörung zum Schutz von Empfängern in Fahrzeugen, Booten und Geräten - Grenzwerte und Messverfahren, 2003.

[DN00] M. Di Natale. Scheduling the can bus with earliest deadline techniques. In *Real-Time Systems Symposium, 2000. Proceedings. The 21st IEEE*, pages 259 –268, 2000.

[DRH+08] J. Diaz, E. Rodriguez, L. Hurtado, H. Cacique, N. Vazquez, and A. Ramirez. Can bus embedded system for lighting network applications. In *Circuits and Systems, 2008. MWSCAS 2008. 51st Midwest Symposium on*, pages 531 –534, 2008.

[DRPN07] J. Degesys, I. Rose, A. Patel, and R. Nagpal. Desync: self-organizing desynchronization and tdma on wireless sensor networks. In *Proceedings of the 6th international conference on Information processing in sensor networks*, pages 11–20. ACM, 2007.

[DZDN+07] A. Davare, Qi Zhu, M. Di Natale, C. Pinello, S. Kanajan, and A. Sangiovanni-Vincentelli. Period optimization for hard real-time distributed automotive systems. pages 278–283, June 2007.

[ESAT06] Tamer ElBatt, Cem Saraydar, Michael Ames, and Timothy Talty. Potential for intra-vehicle wireless automotive sensor networks. In *Sarnoff Symposium, 2006 IEEE*, pages 1 –4, 2006.

[Fal04] O. Falkner. The ideal Tool Chain. for LIN and CAN. From design to hardware and software integration. *Elektronik Automotive, LIN Special*, 2004.

[FB04] Zuyuan Fang and B. Bensaou. Fair Bandwidth Sharing Algorithms based on Game Theory Frameworks for Wireless Ad-hoc Networks. In *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, pages 1284–1295, 2004.

[FC09] FlexRay Consortium. FlexRay Communications Systems - Protocol Specification v3.0. `http://www.flexray.com`, 2009.

[Fel05] M. Felser. Real-time ethernet - industry prospective. *Proceedings of the IEEE*, 93(6):1118 –1129, 2005.

[FL98] D. Fudenberg and D.K. Levine. *The theory of learning in games*. The MIT press, 1998.

[Flo01] S. Floyd. A report on recent developments in tcp congestion control. *Communications Magazine, IEEE*, 39(4):84 –90, apr 2001.

[FMN07] Amos Fiat, Yishay Mansour, and Uri Nadav. Efficient contention resolution protocols for selfish agents. In *SODA '07: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 179–188, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.

[GCGCF08] F. Gil-Castineira, F.J. Gonzalez-Castano, and L. Franck. Extending vehicular can fieldbuses with delay-tolerant networks. *Industrial Electronics, IEEE Transactions on*, 55(9):3307 –3314, 2008.

[Ger05] C. Gershenson. A general methodology for designing self-organizing systems. *Arxiv preprint nlin/0505009*, 2005.

[GGCG10] F. Galea, E. Gatt, O. Casha, and I. Grech. Control unit for a continuous variable transmission for use in an electric car. In *Electronics, Circuits, and Systems (ICECS), 17th IEEE International Conference on*, pages 247–250, dec. 2010.

[GGN+06] M. Grenier, J. Goossens, N. Navet, et al. Near-optimal fixed priority preemptive scheduling of offset free systems. In *14th International Conference on Real-time and Network Systems*. Citeseer, 2006.

[GHN08] M. Grenier, L. Havet, and N. Navet. Pushing the limits of CAN-scheduling frames with offsets provides a major performance boost. In *Proc. of the 4th European Congress Embedded Real Time Software (ERTS 2008), Toulouse, France*, 2008.

[GNW+06] M. Gabrick, R. Nicholson, F. Winters, B. Young, and J. Patton. Fpga considerations for automotive applications. In *Proc. SAE Conf*, 2006.

[Goe97] Jasper J. Goedbloed. *EMV - elektromagnetische Verträglichkeit*. Pflaum, Munich, 1997.

[Goo03] J. Goossens. Scheduling of offset free systems. *Real-Time Systems*, 24(2):239–258, 2003.

[GRS+96] L. George, N. Rivierre, M. Spuri, et al. Preemptive and non-preemptive real-time uniprocessor scheduling. 1996.

[GSFR03] J. Gamiz, J. Samitier, JM Fuertes, and O. Rubies. Practical evaluation of messages latencies in CAN. In *Emerging Technologies and Factory Automation, 2003. Proceedings. ETFA'03. IEEE Conference*, pages 185–192, 2003.

[GT06] T. Gandhi and M.M. Trivedi. Vehicle surround capture: Survey of techniques and a novel omni-video-based approach for dynamic panoramic surround maps. *Intelligent Transportation Systems, IEEE Transactions on*, 7(3):293 –308, 2006.

[Hei] Heidelberger Druckmaschinen AG.

[HHJ+05] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst. System level performance analysis-the SymTA/S approach. In *Computers and Digital Techniques, IEE Proceedings-*, volume 152, pages 148–166. Iet, 2005.

[HHKG09] T. Herpel, K.S. Hielscher, U. Klehmet, and R. German. Stochastic and deterministic performance evaluation of automotive CAN communication. *Computer Networks*, 53(8):1171–1185, 2009.

[HR95] M. Hamdaoui and P. Ramanathan. A dynamic priority assignment technique for streams with (m, k)-firm deadlines. *Computers, IEEE Transactions on*, 44(12):1443 –1451, dec 1995.

[HRGD05] Martin Heusse, Franck Rousseau, Romaric Guillier, and Andrzej Duda. Idle Sense: An Optimal Access Method for High Throughput and Fairness in Rate Diverse Wireless LANs. In *SIGCOMM*, pages 121–132, 2005.

[iA] CAN in Automation. Canopen. `http://www.canopen.org/`.

[Inf08] Infineon Technologies AG. *TLE6250 High Speed CAN-Transceiver*, 2008.

[JCH84] R. Jain, D.M. Chiu, and W.R. Hawe. *A quantitative measure of fairness and discrimination for resource allocation in shared computer system.* Tech. Rep. DEC-TR-301, Digital Equipment Corporation, 1984.

[jta01] Ieee standard test access port and boundary-scan architecture. *IEEE Std 1149.1-2001*, pages i –200, 2001.

[KC03] J.O. Kephart and D.M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[KG93] H. Kopetz and G. Grunsteidl. Ttp - a time-triggered protocol for fault-tolerant real-time systems. In *Fault-Tolerant Computing, 1993. FTCS-23. Digest of Papers., The Twenty-Third International Symposium on*, pages 524 –533, June 1993.

[KHHG08] U. Klehmet, T. Herpel, K.-S. Hielscher, and R. German. Real-time guarantees for can traffic. In *Vehicular Technology Conference, 2008. VTC Spring 2008. IEEE*, pages 3037 –3041, may 2008.

[Kop11] H. Kopetz. *Real-time systems: design principles for distributed embedded applications*, volume 25. Springer-Verlag New York Inc, 2011.

[KSF+09] J. Keinert, T. Schlichter, J. Falk, J. Gladigau, C. Haubelt, J. Teich, M. Meredith, et al. Systemcodesigner - an automatic esl synthesis approach by design space exploration and behavioral synthesis for streaming applications. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 14(1):1, 2009.

[KT75] L. Kleinrock and F. Tobagi. Packet switching in radio channels: Part i–carrier sense multiple-access modes and their throughput-delay characteristics. *Communications, IEEE Transactions on*, 23(12):1400 – 1416, dec 1975.

[LA03] D. Lee and G. Allan. Fault-tolerant clock synchronisation with microsecond-precision for can networked systems. In *Proceedings of the 9th International CAN Conference, Munich, Germany*, 2003.

[LD06] N. Lavesson and P. Davidsson. Quantifying the impact of learning algorithm parameter tuning. In *In Proc. of the National Conference on Artificial Intelligence*, volume 21, page 395. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006.

[LHR+05] G. Lipsa, A. Herkersdorf, W. Rosenstiel, O. Bringmann, and W. Stechele. Towards a framework and a design methodology for autonomic soc. In *Autonomic Computing, 2005. ICAC 2005.*

Proceedings. Second International Conference on, pages 391–392. IEEE, 2005.

[LIN98] LINEAR Technology Corporation. *Datasheet LT1720/LT1721*, 1998.

[LIN01] LINEAR Technology Corporation. *LT1796 - Overvoltage Fault Protected CAN Transceiver*, 2001.

[Lit01] M.L. Littman. Value-function reinforcement learning in Markov games. *Cognitive Systems Research*, 2(1):55–66, 2001.

[LPY97] K.G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a Nutshell. *International Journal on Software Tools for Technology Transfer (STTT)*, 1(1):134–152, 1997.

[LR00] M. Lauer and M. Riedmiller. An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In *In Proc. 17th International Conf. on Machine Learning*, 2000.

[Lun05] J. Lunze. *Regelungstechnik 1: systemtheoretische grundlagen, analyse und entwurf einschleifiger regelungen*, volume 1. Springer DE, 2005.

[MMS06] M. Mnif and C. Muller-Schloer. Quantitative emergence. In *Adaptive and Learning Systems, 2006 IEEE Mountain Workshop on*, pages 78–84. IEEE, 2006.

[MMTS96] H. Mori, Y. Mano, H. Takada, and K. Sakamura. μitron bus: a real-time control lan for open network environment. In *Real-Time Computing Systems and Applications, 1996. Proceedings., Third International Workshop on*, pages 227 –234, oct-1 nov 1996.

[MNB11] A. Monot, N. Navet, and B. Bavoux. Impact of clock drifts on can frame response time distributions. In *Emerging Technologies Factory Automation (ETFA), 2011 IEEE 16th Conference on*, pages 1 –4, sept. 2011.

[Moh] Igor Mohor. Can protocol controller. `http://opencores.org/project,can`.

[MPDO09] P. Marino, F. Poza, M.A. Dominguez, and S. Otero. Electronics in automotive engineering: A top-down approach for implementing industrial fieldbus technologies in city buses and coaches. *Industrial Electronics, IEEE Transactions on*, 56(2):589 –600, 2009.

[MTAB07] Mateusz Majer, Jürgen Teich, Ali Ahmadinia, and Christophe Bobda. The Erlangen Slot Machine: A Dynamically Reconfigurable FPGA-based Computer. *J. VLSI Signal Process. Syst.*, 47(1):15–31, 2007.

[Nas51] John Nash. Non-Cooperative Games. *The Annals of Mathematics, Second Series*, 54(2):286 – 295, 1951.

[NHNP01] T. Nolte, H. Hansson, C. Norström, and S. Punnekkat. Using bit-stuffing distributions in can analysis. In *IEEE Real-Time Embedded Systems Workshop at the Real-Time Systems Symposium*. Citeseer, 2001.

[NNH05] T. Nolte, M. Nolin, and H.A. Hansson. Real-time server-based communication with can. *Industrial Informatics, IEEE Transactions on*, 1(3):192 – 201, 2005.

[NP12] N. Navet and H. Perrault. Can in automotive applications: a look forward. In *13th International CAN Conference, Hambach Castle*, 2012.

[NSL09] N. Navet and F. Simonot-Lion. *Automotive embedded systems handbook*. CRC, 2009.

[NSSE10] Moritz Neukirchner, Steffen Stein, Harald Schrom, and Rolf Ernst. A software update service with self-protection capabilities. In *Proc. of Design, Automation, and Test in Europe (DATE)*, Dresden, Germany, mar 2010.

[NXP00] NXP. *Datasheet SJA1000 Stand-alone CAN controller*, 2000.

[Par07] Moonju Park. Non-preemptive fixed priority scheduling of hard real-time periodic tasks. In *Computational Science - ICCS 2007*, Lecture Notes in Computer Science, pages 881–888. Springer Berlin / Heidelberg, 2007.

[PAT07] N. Pereira, B. Andersson, and E. Tovar. Widom: A dominance protocol for wireless medium access. *Industrial Informatics, IEEE Transactions on*, 3(2):120–130, 2007.

[PC07] C.E. Pereira and L. Carro. Distributed real-time embedded systems: Recent advances, future trends and their impact on manufacturing plant control. *Annual Reviews in Control*, 31(1):81–92, 2007.

[PEPP04] P. Pop, P. Eles, Z. Peng, and T. Pop. Analysis and optimization of distributed real-time embedded systems. In *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, volume 11, pages 593–625. ACM, 2004.

[Per10] N.A.M. Pereira. *Efficient aggregate computations in large-scale dense wireless sensor networks*. PhD thesis, Universidade do Minho, 1010.

[PPA+09] J. Pimentel, J. Proenza, L. Almeida, G. Rodríguez-Navas, M. Barranco, and J. Ferreira. *Automotive Embedded Systems Handbook*, chapter Dependable automotive CAN networks. CRC Press, page 6-5, 2009.

[Rac10] R. Racu. The role of timing analysis in automotive network design. Talk, 4th Symtavision NewsConference on Timing Analysis, Braunschweig, Germany, 2010.

[RG05] Sudipta Rakshit and Ratan K. Guha. Fair Bandwidth Sharing in Distributed Systems: A Game-Theoretic Approach. *IEEE Transactions on Computers*, 54(11):1384–1393, 2005.

[RMB+06] U. Richter, M. Mnif, J. Branke, C. Müller-Schloer, and H. Schmeck. Towards a generic observer/controller architecture for organic computing. *Informatik*, pages 112–119, 2006.

[RS09] RTaW-Sim. Real-time at Work CAN Simulator. `http://www.realtimeatwork.com/software/rtaw-sim/`, 2009.

[Sak94] K. Sakamura. After a decade of tron, what comes next? In *TRON Project International Symposium, 1994., Proceedings of the 11th*, pages 2 –16, dec 1994.

[SBF06] J. Sommer, L. Burgstahler, and V. Feil. An analysis of automotive multi-domain can systems. *Proceedings of the 12th EUNICE Open European Summer School*, 2006.

[SBP+09] J. Sudeikat, L. Braubach, A. Pokahr, W. Renz, and W. Lamersdorf. Systematically engineering self-organizing systems: The sodekovs approach. *Electronic Communications of the EASST*, 17(0), 2009.

[Sch05] H. Schmeck. Organic computing-a new vision for distributed embedded systems. In *Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, 2005. ISORC 2005*, pages 201–203, 2005.

[SE08] Steffen Stein and Rolf Ernst. Distributed performance control in organic embedded systems. In *IEEE 5th International Conference on Autonomic and Trusted Computing (ATC-08) Autonomic and Trusted Computing (LNCS)*, volume 5060/2008 of *Lecture Notes in Computer Science*, pages 331–342. Springer Berlin / Heidelberg, June 2008.

[See11] Hella Seebach. *Konstruktion selbst-organisierender Softwaresysteme*. PhD thesis, Universität Augsburg, 2011.

[SHS10] I. Sheikh, M. Hanif, and M. Short. Improving information throughput and transmission predictability in controller area networks. In *Industrial Electronics (ISIE), 2010 IEEE International Symposium on*, pages 1736 –1741, july 2010.

[SMSc+10] Hartmut Schmeck, Christian Müller-Schloer, Emre Çakar, Moez Mnif, and Urban Richter. Adaptivity and self-organization in organic computing systems. *ACM Trans. Auton. Adapt. Syst.*, 5:10:1–10:32, September 2010.

[SREP08] S. Samii, S. Rafiliu, P. Eles, and Z. Peng. A simulation methodology for worst-case response time estimation of distributed real-time systems. In *Proceedings of the conference on Design, automation and test in Europe, DATE*, pages 556–561. ACM, 2008.

[SRSB98] J.A. Stankovic, K. Ramamritham, M. Spuri, and G.C. Buttazzo. *Deadline scheduling for real-time systems*. Springer, 1998.

[SSA06] F. Sethna, E. Stipidis, and F.H. Ali. What lessons can controller area networks learn from flexray. In *Vehicle Power and Propulsion Conference*, pages 1 –4, sept. 2006.

[SSY10] I. Sheikh, M. Short, and K. Yahya. Analysis of overclocked controller area network. In *Networked Sensing Systems (INSS), 2010 Seventh International Conference on*, pages 37 –40, june 2010.

[Str07] Thilo Streichert. *Self-Adaptive Hardware/Software Reconfigurable Networks - Concepts, Methods, and Implementation*. PhD thesis, Universität Erlangen-Nürnberg, Universitätsstrasse. 4, 91054 Erlangen, 2007.

[SV07] A. Sangiovanni-Vincentelli. Quo vadis, sld? reasoning about the trends and challenges of system level design. *Proceedings of the IEEE*, 95(3):467–506, 2007.

[TBY+96] J.J.P. Tsai, Y. Bi, S. Yang, R.A.W. Smith, and AW Ross. *Distributed real-time systems: monitoring, visualization, debugging, and analysis*. Wiley, 1996.

[Tei12] J. Teich. Hardware/software codesign: The past, the present, and predicting the future. *Proceedings of the IEEE*, 100(Special Centennial Issue):1411 –1430, 13 2012.

[Tex03] Texas Instruments Incorporated. *Datasheet SN54AHCT125, SN74AHCT125 Quadruple bus buffer gates with 3-state outputs*, 2003.

[THW95] Ken Tindell, Hans Hansson, and Andy Wellings. Analysing Real-Time Communications: Controller Area Network (CAN). In *Proc. 15th IEEE Real-Time Systems Symposium*, San Juan, Puerto Rico, December 1995. IEEE Society Press.

[WD92] C.J.C.H. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992.

[Xil04] Xilinx. *Digital Clock Manager (DCM) Module*, 2004.

[ZLH08] F. Zhou, S. Li, and X. Hou. Development method of simulation and test system for vehicle body CAN bus based on CANoe. In *7th World Congress on Intelligent Control and Automation, WCICA*, pages 7515–7519. IEEE, 2008.

[ZS95] K.M. Zuberi and K.G. Shin. Non-preemptive scheduling of messages on controller area network for real-time control applications. In *Real-Time Technology and Applications Symposium, 1995. Proceedings*, pages 240 –249, may 1995.

Author's Own Publications

[WZT09*] Stefan Wildermann, Tobias Ziermann, and Jürgen Teich. Self-organizing bandwidth sharing in priority-based medium access. In *Proceedings of the Third IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 09)*, pages 144–153, San Francisco, USA, 2009.

[ZBZT12*] Tobias Ziermann, Alexander Butiu, Daniel Ziener, and Jürgen Teich. Fpga-based testbed for timing behavior evaluation of the controller area network (can). In *Proceedings of the International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, Dec. 2012.

[Zie08*] Tobias Ziermann. Konzepte und umsetzung zeitvarianter can-protokolle durch rekonfigurierbare hardware. Diplomarbeit, Department of Computer Science, University of Erlangen-Nuremberg, 2008.

[ZMWT10*] Tobias Ziermann, Nina Mühleis, Stefan Wildermann, and Jürgen Teich. A self-organizing distributed reinforcement learning algorithm to achieve fair bandwidth allocation for priority-based bus communication. In *1st IEEE Workshop on Self-Organizing Real-Time systems (SORT 2010)*, pages 11–20, Carmona, Spain, May 2010.

[ZST11a*] Tobias Ziermann, Zoran Salcic, and Jürgen Teich. DynOAA - dynamic offset adaptation algorithm for improving response times of CAN systems. In *Proc. of Design, Automation, and Test in Europe (DATE)*, pages 269–272, 2011.

[ZST11b*] Tobias Ziermann, Zoran Salcic, and Jürgen Teich. Self-organized message scheduling for asynchronous distributed embedded systems. In *Proceedings of 8th International Conference on Autonomic and Trusted Computing*, pages 132–148, 2011.

[ZST12*] Tobias Ziermann, Zoran Salcic, and Jürgen Teich. Improving Performance of Controller Area Network (CAN) by Adaptive Message Scheduling. In M. Teresa Higuera-Toledano, Uwe Brinkschulte, and Achim Rettberg, editors, *Distributed, Embedded, and Real-Time Systems*, pages 95–120. Springer, Heidelberg, 2012.

[ZST13*] Tobias Ziermann, Zoran Salcic, and Jürgen Teich. Hw/sw tradeoffs for dynamic message scheduling in controller area network (can). In *Proceedings of the International Conference on Architecture of Computing Systems (ARCS)*, Prague, Czech Republic, February 2013.

[ZT10a*] Tobias Ziermann and Jürgen Teich. Adaptive Traffic Scheduling Techniques for Mixed Real-Time and Streaming Applications on Reconfigurable Hardware. In *Proceedings of 17th Reconfigurable Architectures Workshop (RAW)*, pages 1 –4, 2010.

[ZT10b*] Tobias Ziermann and Jürgen Teich. Electromagnetic Compatibility (EMC) of CAN+. In *Proc. of Automotive meets Electronics (AmE 2010)*, pages 25–30, 2010.

[ZWMT11*] Tobias Ziermann, Stefan Wildermann, Nina Mühleis, and Jürgen Teich. Distributed self-organizing bandwidth allocation for priority-based bus communication. *Concurrency and Computation: Practice and Experience*, 2011.

[ZWT09a*] Tobias Ziermann, Stefan Wildermann, and Jürgen Teich. Can+: A new backward-compatible controller area network (can) protocol with up to 16x higher data rates. In *Design, Automation Test in Europe Conference Exhibition, DATE '09*, pages 1088 –1093, april 2009.

[ZWT09b*] Tobias Ziermann, Stefan Wildermann, and Jürgen Teich. CAN+: Techniques and prototype for achieving increased data rates on the basis of common CAN bus structures. In *Proceedings of 9th Stuttgart, International Symposium*, pages 327–339, Stuttgart, Germany, March 2009. ATZlive/GWV Fachverlage GmbH.

[ZWT11*] Tobias Ziermann, Stefan Wildermann, and Jürgen Teich. Organicbus: Organic self-organising bus-based communication systems. In Christian Müller-Schloer, Hartmut Schmeck, and Theo Ungerer, editors, *Organic Computing – A Paradigm Shift for Complex Systems*, volume 1 of *Autonomic Systems*, pages 489–501. Springer Basel, 2011.

List of Symbols

α	fair bandwidth
Δ	unweighted amount of adaptation
ϵ	very small number
ζ	overclocking factor
η	learning rate
θ	hardware PPLA parameter
λ	wavelength
μ	accuracy
σ	source bus of streams
ψ_{routed}	percentage of routed messages
$\psi_{received}$	percentage of received segments
ω	destination/sink bus of streams
A	set of strategies
a	strategy
$\mathbf{a}$	vector of strategies
B	blocking for Tindell analysis
b	bandwidth
c	speed of light
d	number of buses
F	utility function
f	frequency
G	game
G_{mixed}	mixed game
I	interference for Tindell analysis
i	index
j	index
K	set of player
k	number of players
k_{add}	number of additional players
l	length of a message
l_{mi}	length of monitoring interval
l_c	length of cable
m	message

m_{sent}	number of messages sent
N	node
n	number of streams
O	set of offsets of streams
P	hyper period
p	sending probability
$\mathbf{p}$	vector of probabilities
p_{free}	probability that medium is free
p_i^{start}	starting probability of player i
q	number of memory locations
R	requirement type of a stream (hard/soft deadline, bandwidth)
r	bit rate
s	message stream
s'	multi-segement message stream
$success$	percentage of successful transmitted messages
$nSuccess$	number of successful transmitted messages
T	period of streams
T^{start}	initial period
T_{fair}	fair period
t	time step
t_a	beginning of WCRT intervall
t_b	end of WCRT intervall
$t_{converge}$	convergence time
t_{sim}	simulation time
t_{sample}	relative time of the sample point
t_{bit}	length of a bit
t_{busy}	amount of time busy
t_{idle}	amount of time idle
t_s	start of the gray zone
t_e	end of the gray zone
$\hat{t}_{bit}$	length of an overclocked bit
U	occurence of a stream
$WCRT$	worst case response time
w	workload

Acronyms

AWW	Average Weighted WCRT
CAN	Controller Area Network
DES	Distributed Embedded Systems
DPR	dynamic partial reconfiguration
DynOAA	Dynamic Offset Adaptation Algorithm
ECU	Electronic Control Unit
FPGA	Field Programmable Gate Array
fps	frames per second
FSM	Finite State Machine
IC	integrated circuit
LIT	longest idle time
LBT	longest busy time
PLA	Penalty Learning Algorithm
PPLA	Period Penalty Learning Algorithm
staticOAA	Static Offset Assignment Algorithm
TQ	Time Quantum
WSN	Wireless Sensor Network
WCRT	Worst-Case Response Time

Curriculum Vitæ

Tobias Ziermann received his masters degree (Dipl.) in computer science at the University of Würzburg in 2008. Since 2008, he is working as a PhD student in the research group of Prof. Jürgen Teich at the Department of Computer Science at the University of Erlangen-Nuremberg.

His research interests include adaptive communication of distributed embedded systems, learning algorithms and design of FPGA cores.