Thesis for the degree of Doctor of Philosophy

# Efficient and Reliable Communication in Distributed Embedded Systems

VILGOT CLAESSON


Department of Computer Engineering
School of Computer Science and Engineering
Chalmers University of Technology
Göteborg, Sweden 2002

**Efficient and Reliable Communication in Distributed Embedded Systems**
Vilgot Claesson
ISBN 91-7291-234-0



Doktorsavhandlingar vid Chalmers tekniska högskola
Ny serie 1916
ISSN 0346-718X

School of Computer Science and Engineering
Chalmers University of Technology
Technical report No 6D
ISSN 1651-4971

Department of Computer Engineering
Chalmers University of Technology
SE–412 96 Göteborg, Sweden
Phone: +46 (0)31–772 1000
www.ce.chalmers.se

Author email address: `vilgotc@ce.chalmers.se`

Vasastadens Bokbinderi
Göteborg, Sweden 2002

# Efficient and Reliable Communication in Distributed Embedded Systems

Vilgot Claesson
*Department of Computer Engineering, Chalmers University of Technology*

## Abstract

The use of distributed computing elements has grown in the embedded systems arena, consequently the use of shared communication media linking these computers has garnered increasing attention. Two prominent and contemporary media sharing approaches include variations of *controlled* and *contention* based media access paradigms, with the time-triggered (TT) and the event-triggered (ET) approaches, respectively, being prominent manifestations of these paradigms. For these mentioned TT and ET approaches, the thrust of this thesis is on investigating and analyzing efficient realizations of such.

A desired attribute in embedded systems, with safety and real-time requirements, is the basic capability to coordinate and synchronize system time and events. This directly relates to the establishment of a predictable communication base, which subsequently becomes the basis to provide for predictable communication at the system level. Focusing on bus-based communication protocols, we present a novel synchronization approach targeting efficiency and low communication overhead as the main drivers for TDMA environments. Existing techniques require explicit transfer of node *ID* information for synchronization. In this novel approach, the synchronization process utilizes the implicit information in each node's unique message length as node identifier. Furthermore, our initial startup synchronization approach is fault-tolerant, and has a bounded startup time. We also present a re-synchronization strategy that incorporates recovering nodes into synchronization.

The event-triggered and the time-triggered media access paradigms, have spawned discrete followings with much debated pros and cons regarding their relative flexibility, bandwidth efficiency and predictability features. The event-triggered approach is commonly perceived as providing high flexibility. Similarly, the time-triggered approach is expected to provide a higher degree of predictable communication access to the media. One part of this thesis is to objectively and quantitatively assess the capabilities and limitations of each of these paradigms. More importantly, we quantify the spread of their differences, and provide system design guidelines for suggested best usage for each approach. The focus of this work is on response times of the communication system, and the schedulability

of the communication system in collaboration with tasks in the nodes. Focusing on efficiency, the second component of this thesis, deals with introducing modifications in the time-triggered approach to efficiently accommodate event-triggered communication using the time-triggered operations as a base. **Keywords:** distributed embedded systems, synchronization, media access protocols, time-triggered, event-triggered

# Acknowledgments

I consider myself very fortunate that Professor Neeraj Suri and his family decided to move to Sweden (at least for a while) and that he agreed to be my advisor. His support and encouragement has been invaluable for this work and without it, I would not have come this far. He has also created a stimulating, friendly (and mischievous ;-) environment, with lots of social activities.

**THANKS Neeraj!**

I also want to thank a number of other persons and groups who in some way or the other have contributed to this work.

The members of the DEEDS group. Ph.D. Martin Hiller, Örjan Askerdal, Arshad Jhumka, Robert Lindström, and Andreas Johansson. It has been a true pleasure working and socializing with you all. And many extra thanks to Martin, Arshad and Örjan for enduring a lot of proof reading of my material.

My friend and colleague Henrik Lönn whom especially during my first years provided encouragement and assistance.

The members of the old "Bilgruppen" Professor Jan Torin, Kristina Ahlström, Örjan Askerdal, Olle Bridal, Arne Dahlberg, Lars-Åke Johansson, Roger Johansson, Håkan Sivencrona and Rolf Snedsbøl,

People I have had the true pleasure of writing papers with; Lic. Eng. Cecilia Ekelin, Dr. Stefan Poledna and Mr. Jan Söderberg.

All the people I work and worked with in the projects NextTTA, DICOSMOS, and X-By-Wire. I especially have to mention the members of the DICOSMOS Ph.D. Student group: Magnus Gäfvert and Martin Sanfridsson. I will not forget all nice Thai lunches/dinners! ;-)

Rolf Snedsbøl and Arne Linde, it has been really nice working with you, during my teaching obligations.

The students at the department for creating such a cheerful and friendly environment. I also want to thank our support staff for their kind reception and help, i.e., Ewa Cederheim-Wäingelin, Kerstin Germundsson, Per Waborg, Marie Carlsson. Likewise, I want to thank the computer wizards Peter Helander and Christian Roth.

My dear parents Ingrid and Carl-Eric thanks your for all support and love. My sister Annacarin with family Henrik, Amanda, and Marcus.
I am so lucky to have you all!!!

And, Margareta **THANK YOU !**

# Projects

1. **NEXT TTA** - High Confidence Architecture for Distributed Control Applications
   IST-2001-32111
   The objective of NEXT TTA project is to further improve the dependability and functionality of the time-triggered architecture (TTA).

2. **SAAB Professorship, Endorsement**:
   Support of Research: In Dependable and Robust Real-Time Systems.

3. **Volvo Research Foundation Fellowship**
   *Project title:* Communication Services for Safety-Critical Distributed Real-Time Systems. #F99/07
   The author applied and received this competitive grant, for the period 2000-2001.

4. **DICOSMOS2** - Integrated Real-time Computer and Control System Architectures. Financed by NUTEK (the Swedish National Board for Industrial and Technical Development) and by VOLVO. "Projektnr. P11762-2, Dossie-Diarienr. 1K1P-99-06187".
   The author was local manager for DICOSMOS2.
   The goal of the project is to combine industrial and university knowledge of requirements and design of safety critical, real-time, and control systems. This has been achieved using a case study. The area has been investigated and researched with an interdisciplinary perspective, provided by the composition of the group, i.e., control theory, mecatronics, and computer engineering.

5. **DICOSMOS**. The predecessor of DICOSMOS2 where the focus were on reliable distributed control by interdisciplinary work.

6. **Pålbus**.
   The project was financed by NUTEK and by the participating industries. The objective of the PÅLBUS project [Pol99] was to evaluate and disseminate methods for design and evaluation of safety and reliable communication buses, used in distributed control.

7. **X-By-Wire** - Safety Related Fault Tolerant Systems in Vehicles
   European project: Brite-Euram III, project no. BE 95/1329. [X-B98]

The objective of this project was to achieve a framework for the introduction of safety related fault tolerant electronic systems without mechanical or hydraulic backup in vehicles.

## List of Papers and Reports

*This thesis is primarily based on the author's following work.*

1. V. Claesson, H. Lönn, and N. Suri, "*Efficient TDMA Synchronization for Distributed Embedded Systems,*" presented at 20th Symposium on Reliable Distributed Systems (SRDS), New Orleans, USA, 2001.

2. V. Claesson, H. Lönn, and N. Suri, "Efficient TDMA Synchronization for Distributed Embedded Systems" Extended version of paper (1) submitted to IEEE Transaction on Parallel and Distributed Systems, 2002.

3. V. Claesson, S. Poledna, and J. Söderberg, "*The XBW Model for Dependable Real-Time Systems,*" presented at International Conference on Parallel and Distributed Systems, Tainan, Taiwan, 1998.

4. H. Sivencrona, L-Å. Johansson, and V. Claesson, "*A Novel Bit-Oriented Communication Concept for Distributed Real-Time Systems, QRcontrol,*" presented at 3rd International Conference on Control and Diagnostics in Automotive Applications (CDAUTO01), Sestri Levante (Genova), Italy., 2001.

5. V. Claesson and H. Lönn, "*Delay based startup and message length addressing in TDMA communication,*" Chalmers University of Technology, Department of Computer Engineering, Göteborg, Report no. 00-21, 2000.

6. V. Claesson, "*Cost Effective Communication Services for Applications in Distributed Time Triggered Real-Time Systems,*" in Department of Computer Engineering. Göteborg: Chalmers University of Technology, 1999.

7. V. Claesson, "*Atomic Broadcast and Membership Agreement in Time Triggered real-time systems,*" Department of Computer Engineering, Chalmers University of Technology, Gothenburg 99-6, 1999.

8. V. Claesson, C. Ekelin, and N. Suri, "*The Event-Triggered and Time-Triggered Medium-Access Methods*", submitted to The 6th IEEE International Symposium on Object-oriented Real-time distributed Computing.

9. NEXT TTA WP2: participants, "ET & TT Requirements Document" NEXT TTA, IST-2001-32111, WP2 Review Report.

10. V. Claesson, H. Lönn, and N. Suri, "*An Efficient TDMA Synchronization Approach for Distributed Embedded Systems,*" Chalmers University Of Technology, Department of Computer Engineering, Göteborg, Report no. 01-06, 2001.

11. V. Claesson, M. Gäfvert, and M. Sandfridsson, "*Proposal for a distributed computer control system in heavy-duty trucks.,*" Computer Engineering, Chalmers University of Technology., Göteborg, Dicosmos Internal Report 00-16, 2000.

12. M. Sanfridsson, V. Claesson, and M. Gäfvert, "*Investigation and Requirements of a Computer Control System in a Heavy-Duty Truck*," Mechatronics Lab, Royal Institute of Technology., Stockholm, Sweden. TRITA-MMK 2000:5, ISSN 1400-1179, ISRN/MMK–00/5–SE, 2000.

13. Ö. Askerdal, V. Claesson, "*Error Detection and Handling*", Pålbus project task 10.5. 2000.

14. V. Claesson, "*Prototype Implementation Using the XBW Software Model,*" presented at SNART'97, Lund, Sweden, 1997.

## Reports

*Related papers and reports with the author involved.*

1. A. Jhumka, M. Hiller, V. Claesson and N. Suri, "On Systematic Design of Consistent Executable Assertions for Distributed Embedded Software," in Languages, Compilers, and Tools for Embedded Systems (LCTES), 2002.

2. A. Jhumka, M. Hiller, V. Claesson and N. Suri, "On Systematic Design of Consistent Executable Assertions for Distributed Embedded Software," Extended version. Submission: IEEE Transactions on Software Engineering

3. M. Gäfvert, M. Sandfridsson, and V. Claesson "*Truck Model for Yaw and Roll Dynamics Control,*" Technical Report ISRN LUTFD2/TFRT–7588–SE, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden, sept. 2000.

# Contents

# Chapter 1

# Introduction

THE use of computers in embedded systems has been growing for many years, and there is little reason to believe that this trend will stop. On the contrary, complex computers are appearing as integrated components and their usage spreading from provision of support services to primary provision of safety-critical operations in systems ranging from automotive and aerospace to e-applications. Their usage introduces new and improved functionality where computer control is dominant, but usage in other areas is constantly increasing, such as infotainment, active-sensors, and active safety systems.

Increasingly, these systems also tend to become more closely coupled, forming large and closely interacting distributed systems. Thus, large numbers of embedded computers in different sub-systems work together interconnected with one or many communication systems. Consequently, the communication media linking the computing nodes and subsystems becomes a key ingredient in facilitating *system level* efficiency and reliability. This translates to the importance of efficiency and reliability at the *communication media level* itself.

However, most commercial embedded systems are also very cost sensitive, contrary to some of the early systems with extreme requirements on safety and reliability, and with related large budgets, e.g., nuclear and space applications. Although, such systems have been built with success, they are seldom adaptable to this new trend of embedded systems where cost together with efficiency and safety requirements are the major concerns.

The safety requirements of such distributed systems, necessitate reliable communication that generally requires redundancy mechanisms as well as fast error detection and recovery. Furthermore, the hard real-time requirements of many embedded systems introduce the additional demand of timely response. Thus, there is a need for efficient and low

cost communication solutions that can facilitate safety and reliability.

With cost-efficiency, real-time, and fault-tolerance as the main system drivers, there are two communication paradigms of special interest for our work, namely controlled- and contention-based media access. The time-triggered and event-triggered approaches, respectively, are prominent manifestations of these access communication paradigms and constitute our interest in this thesis. We do emphasize that our discussion of time-triggered and event-triggered approaches focuses at the communication access methodology, and not necessarily at the event-triggered and the time-triggered system architecture concepts.

The first is the time-triggered approach, which uses Time Division Multiple Access (TDMA) as media access method. With this method, the media access is divided in time such that each node is assigned a time slot periodically. This means that nodes will send information at predefined times, triggered by a global time base. The use of a time-triggered approach follows a deterministic behavior, which facilitates meeting hard reliability requirements.

The other communication paradigm is the event-triggered approach, which normally uses contention-based communication. This is generally considered a more flexible communication paradigm. However, the real-time properties are very limited in the pure form of contention-based media access method. There are a number of methods to overcome this, which are used in event-triggered systems, e.g., by arbitrating messages accessing the media on the fly.

It is within this area that we have centered our attention throughout the work in this thesis. In the next section, we further describe the assumed system environment and operational conditions where our work best applies.

## 1.1 System Model

In this section, we briefly describe the main system models, concerning both time-triggered and event-triggered, that we use throughout this thesis. In relevant chapters further details will be discussed, and some variations will be described.

We target systems that have stipulated requirements of reliability and real-time. In these systems, we consider $n$ autonomous nodes that communicate via a broadcast media. When using such a media, all nodes can simultaneously listen and receive information transferred on the media. However, only one node can send at a time; if more than one node transmits concurrently, the information will be garbled.

### 1.1.1 Time-Triggered System

In the case of a *time-triggered system*, nodes have local counters that are used to control the sending and receiving of messages. Furthermore, the nodes operate in a cyclic manner where each node can send one or more times in each TDMA communication round, as shown in Figure 1.1. However, for ease of explanation, we will assume that nodes send exactly once in each TDMA round, if nothing else is stated. Each node $i$ may send messages of different lengths denoted by $tm_i$.

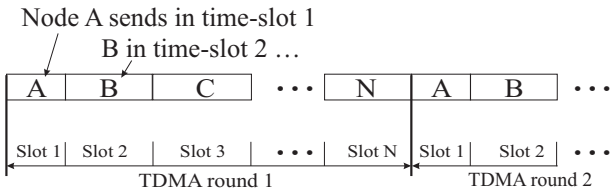


Figure 1.1: A TDMA communication round.

**Synchronization:** We consider synchronization to be divided into two logical levels, (1) synchronization at the frame level and (2) synchronization at the communication schedule level. We assume that the frame level synchronization is handled by a standard clock synchronization algorithm controlling the progress of the local clock, for example, using the daisy-chain clock synchronization algorithm [LS95]. Other variants and further information of clock synchronization issues can also be found in [SHW94]. The frame level synchronization ensures that each bit-value is received correctly.

Synchronization at the communication schedule level synchronizes the nodes to the same phase of the communication schedule. This ensures that every node has the same view of the current position in the communication schedule. Together with synchronized local clocks, this global position knowledge is sufficient for nodes to know when to send their messages. Thus, the necessary information for a node is the global time, which consists of fine-grained time ensuring correct bit values, and the coarse-grained time that corresponds to the position in the communication schedule.

### 1.1.2 Event-Triggered System

In the case of an *event-triggered system*, a node will immediately send its message if the media is free. If the media is occupied, the transmission is delayed until the media becomes free. However, if more than one node is

waiting to send when the media becomes free, a collision is likely to occur. In our work, we have assumed that this contention can be resolved using established contention resolution techniques such as those mentioned in Chapter 1.2. This implies that one node will 'win' and the other nodes will withdraw.

**Synchronization:** In an event-triggered system synchronization is not really necessary, as nodes do not synchronize their sending with other nodes. However, there is nothing that prevents the introduction of synchronization in the event-triggered system. To achieve this, some additional mechanisms are required, but solutions for this exist, such as [LH01].

### 1.1.3 Event-Triggered and Time-Triggered Example Run

In this section we depict visual illustration of some properties of the event-triggered and time-triggered communication via two figures. The first, Figure 1.2 shows an example run of an event-triggered communication, and similarly in Figure 1.3 we show a example run of a time-triggered system. In both figures, the differently shaded gray bars represent message-frames on the communication media, sent by different computer nodes (see the legends). The x-axis depicts progression of time. The y-axis depicts a progression of periods, $T$, of the periodic message transmission by the system nodes. In Figures 1.2 and 1.3, the simulations start at time zero, and outline the traffic of the current sending node.

A frame will contain one or more messages. In the event-triggered case normally only one message is packed in each frame. However, in the time-triggered case, more messages are packed as one node is normally only allowed to send one frame per TDMA-round.

In these two simulations, we generate traffic to fill the media with approximately one third periodic messages. The rest of the media time has been used for sporadic messages. We can see this very easily in the event-triggered case, Figure 1.2, where each node sends only one message per frame. We see how the periodic messages are sent regularly. However, we can also see that some times these periodic message are delayed by some sporadic message. If we, for example, follow the checkered bar, i.e, node 1, we can see that this node has a periodic message scheduled at time $0, T, 2T$ etc. However, as we can see it sometimes gets disturbed by other messages, such that it suffers slight delays. We also show this effect by following the periodic messages form node 2, i.e., the black bars, with a grey line, see Figure 1.2.

In the time-triggered case, Figure 1.3, we can see how each node regularly sends a frame, containing messages. As we can see, there are

no message delays and no conflicts among the sending nodes as they send at predefined points in time. Sporadic messages will be packed in the frames, in parts that are not used for periodic messages.

### 1.1.4 Fault Model

When considering faults, we assume, if nothing else is stated, that nodes follow a *fail-silent* semantics, which, for example, prevents faulty nodes to fail by continuously transmitting on the media. Such a failure would overflow the media and prevent any normal communication, including synchronization traffic. The fail-silent property [CPR+92, KDK+89, Tem98] relies on high coverage of the error detection mechanisms of the nodes. It can be argued that sufficient coverage may be difficult to achieve and therefore, such failure semantics is unsuitable for hard real-time system. However, recent work indicates that using rigorous design and error detection methods, a very high coverage can be achieved [Fol99].

The fail-silent semantics will reduce the number of diverse failure scenarios we need to consider in our system.

#### The Media

For the broadcast media, we assume well-accepted omission failure semantics [PT86, Pow92] where messages are either received correctly and on time or not at all. By designing nodes to be fail-silent and using a broadcast media, we effectively exclude timing failure on the media. Similarly, Byzantine failures are not regarded as they are considered to be avoided by design, using a broadcast media combined with message checksums.

## 1.2 Multiple Access Approaches

IN this section, we will give a background and short history to multiple-access communication; we do not, however, claim to cover every aspect of this subject. For further information, we direct the reader to [KSY84] and [MZ95] which survey the area.

The demand for multiple access networks started very early when there was a demand for communication among different computers. Using the same media, reduced the number of cables required significantly and became an inexpensive and efficient solution. The multiple access communication method has since been used extensively. The most well known use of this approach is Ethernet [MB76].

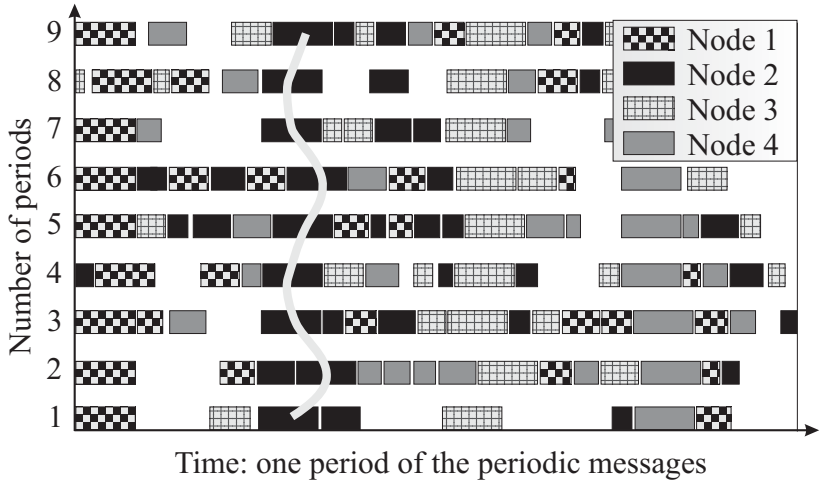


Figure 1.2: An event-triggered communication example run. The vertical wave-like gray line follows the periodic messages from node 2.

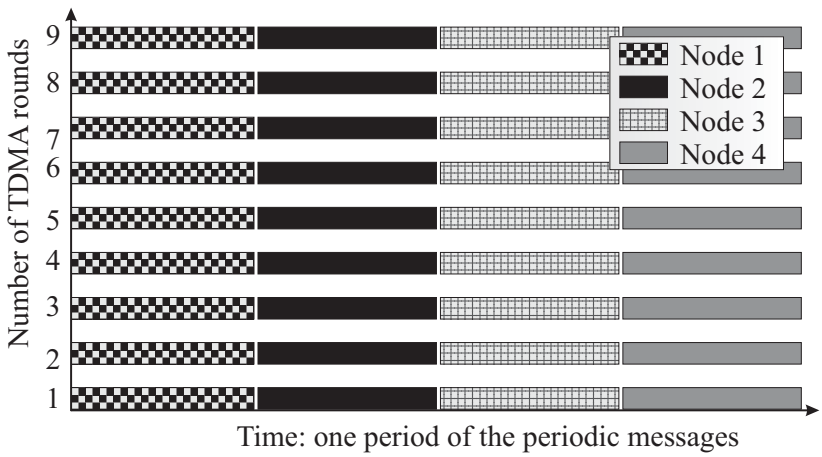


Figure 1.3: A time-triggered communication example run.

One basic problem with communication using a shared media is how nodes should avoid colliding when they start sending messages on the shared media. Computer nodes must share the media like people share the "air" during a conversation, there must be an understanding how to avoid and/or resolve collisions.

There exists a number of different protocols for nodes to send on a multiple access media, and there are many ways they can be classified. We have divided the protocols into two major groups, contention based and controlled access protocols, as done in [KSY84]. By controlled access protocols, we refer to protocols that coordinate their access to the media, such that collisions are avoided. In contention-based protocols, there is no mechanism for avoiding collisions, instead they concentrate on sending messages such that the probability of collision is minimized.

### 1.2.1 Control access protocols

Controlled access protocols coordinate the media access such that no collision occurs. This can be done either in a static manner or dynamically during runtime. In the static case, the computer node's media access is coordinated pre-runtime. Thus, each node will be assigned a time window when the node can send messages, these time windows must be repeated in a cyclic manner, see Figure 1.4. Thus, the access time is divided between the nodes, such that all nodes repeatedly get access to the media. This method is therefore called Time Division Multiple Access (TDMA), which we have described previously.

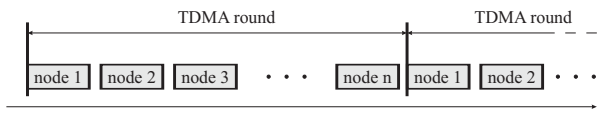


Figure 1.4: Time Division Multiple Access (TDMA) as controlled access protocol.

In order for this access protocol to work, the computer nodes must have a common knowledge of time, which means that they need to be synchronized.

The other type of controlled access protocols is based on dynamic control of the media. This implies that during runtime, nodes must exchange information for coordination of the media access. This information can be sent explicitly or implicitly.

For example, a node can explicitly send a bit indicating the intention to send a message , see [KS80]. The time each node sends its indication,

is statically determined pre-runtime. When all nodes have indicated their send request, these nodes send in a predefined order, e.g., according to the node IDs. When these nodes have sent there messages, this behavior is repeated, starting with a new period where node announce their send intentions.

A similar approach can be achieved by using implicit information, each node are assigned a very short and equal length time slot, i.e., a mini-slot, see Figure 1.5. We can also view these as timeouts, i.e., small mini-slot times. If a node wants to send, it start its message transfer within this slot. Assume a node $i$ is assigned mini-slot number $x$; after a silence on the media for $x$ mini-slots, the node $i$ can access the media. However, $i$ sends its full message directly, occupying more than its mini-slot. After the message is sent, nodes will continue to count mini-slots, i.e., $x + 1$ etc. If node $i$ refrains from sending in its mini-slot, the next mini-slot continues, which gives the access-right to the next mini-slot owner. A longer timeout is used to indicate when a round is finished.

Thus, instead of using these mini time-slots for send-request, the full message is started during these slots. This method is often called mini-slotting or as in [KS80] Minislotted Alternating Priorities (MSAP). As we can see in Figure 1.5, this method can achieve both cycles with fixed length, like TDMA, and cycles with varying length. This depends on how the nodes send there messages. Messages has to be scheduled pre-runtime to get fixed length cycles, i.e., the same amount of data must be transferred each cycle.

The minislotting approach can be found, for example, in the Arinc 629 protocol [ARI95] which is used in Boeing 777 [Yeh96]. Another examples is the byteflight protocol [BPG, Byt99], as well as the FlexRay protocol [BES+01] which uses a slight variation of minislotting.

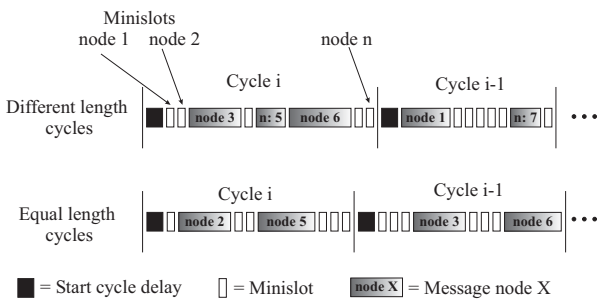


Figure 1.5: Controlled access by mini-slotting.

Another dynamic approach is based on passing a token among the

nodes in a virtual ring. Only the token holder are allowed to access the media. When a node has finished sending, it passes the token to the next node. These are all well known protocols and often referred to as token protocols. In these protocols, nodes are normally assigned a limited time for which they are allowed to hold the token, i.e., a limited time for message transfer. This ensures that all nodes are allowed to send within certain intervals. The exception is when the token is lost. This can happen if a message carrying the token is lost or the node holding the token fails. It is normally quite time consuming for the system to detect and recover a lost token, and a separate recovery protocol is required for this purpose. This is often held against these types of protocols especially in real-time system.

### 1.2.2 Contention Based Protocols

This type of media access uses no distributed protocol to avoid collision. In its simplest form, a node transfers a message when it has information to send. The name 'contention-based' originates from the fact that contentions occur when two or more access the media simultaneously. That is, without any control access method, a collision may occur at any time during the message transfer and the collision will effectively prevent at least two messages from being delivered correctly. Allowing nodes to access the media without any restrictions is not very efficient, but there has been a lot of research, improvements and variants of this branch of media access.

A major step in this development was when nodes were provided with functionality to sense the media, i.e., sense if the media is free or not. After a node starts sending, no collisions will occur as nodes will detect the transmission on the media. This method is called Carrier Sense Multiple-Access (CSMA). However, when two or more nodes begin their transmission at the same time, we still get a contention. This can also be detected by the nodes, by listening to their own transmission. In case of a contention, the transferred data is not the same as the received. The colliding nodes can then back off and re-send the message. Starting this re-send immediately will most likely lead to another collision. This is commonly solved by a random backup time before the node tries a retransmission.

Numerous methods have been developed to improve the contention-based methods. One direction is focusing on minimizing the risk for collisions when nodes start to send. For example, there is the *p-persistent* CSMA, where a node detecting a free media will start to send with a probability *p*. With probability *1 - p* the node will wait some predefined

time, before the same behavior repeats itself. Ethernet [MB76] is a special case of the *p-persistent* CSMA where *p=1*.

Another example builds on dividing the computer nodes into a number of sets. Only one of these sets, say set $A$, is allowed to send. If a collision occurs, the nodes in set $A$ are in turn divided such that, e.g., half remains in $A$ and the rest are removed from $A$ and are not allowed to send. If an additional collision occur, the $A$ set is divided further and the nodes with sending permissions is decreased. As long as collisions occurs, this is repeated and eventually the collisions cease when only one node remains in set $A$. There is a number of ways these sets can be partitioned and divided, but we will not discuss them in detail. To give one example, time can be used to decide these sets, as in the time window protocol, where messages generated within a specific time window are allowed to send. If there are more then one node in this window, a collision will occur. The window will then be decreased to, say half the size such that the nodes with message generated within this time frame are allowed to send. This is repeated until only one node remains. Another way is to use the node ID to divide the nodes into the different sets.

Another direction in contention-based protocols is avoiding collisions by bit arbitration. Each message starts with a priority field, such that nodes can arbitrate which node has the highest priority. This requires nodes to send dominant and recessive bits. This allows a node $a$ sending a recessive bit, to sense and withdraw when a dominant bit is received. When nodes send simultaneously, only the highest priority message will access the media. This is an approach used in, for example, than CAN protocol [CAN91].

In this section we have looked at some of the variations of multiple access approaches. With the background of this section and the system model described, we will in more detail describe the problems we have focused on in this thesis.

## 1.3 Goals and Problem Statements

With this background, overall, our goal has been the achievement of cost and performance efficiency in the described communication systems, but with maintained safety and reliability. More specifically we have targeted the following problems, described in the subsequently mentioned three problem statements [PS1, PS2 and PS3] and elaborated over the thesis chapters:

**PS1** *How to achieve efficient start-up and restart synchronization of TDMA communication?* [ *Chapter 2* ].

Using TDMA in a communications system requires that all nodes have a global opinion of time. This is necessary for a node to know when to send its messages and when to receive messages of interest to it. Thus, by the global time a node can both identify senders and when to send. However, at startup there must be a method to establish and agree on the global time. This requires that messages can be identified without a global time, and time values can be exchanged among the nodes. Similarly, when a node wants to reintegrate after a perturbation, this node must be able to gather the global time from the information transferred on the communication media. Our interest has been how to efficiently establish the global time during startup and reintegration.

**PS2** *What do we gain or lose by choosing the event-triggered vs. time-triggered communication paradigms?* [ *Chapter 3* ].

The event-triggered and time-triggered communication paradigms and their media access methods have in a way been opposite poles in the community. The prevailing opinions are that:

- The event-triggered approach provides flexibility and is efficient in using the communication media.
- The time-triggered approach is more restrictive in how the media is used. Instead it provides a predictability that facilitates reliable designs.

Our interest has been to objectively and quantitatively ascertain the conditions and design space requirements where these statements apply as well as how much difference in efficiency there is under varied communication scenarios.

**PS3** *Can we combine the best of the event-triggered and the time-triggered worlds?* [ *Chapter 4* ].

Finally, we have been interested in if, and in such cases how, we can achieve the best of both worlds, i.e., can we combine the event-triggered and the time-triggered paradigms and keep their best properties. Is it possible to achieve a composite communication system with the predictability of a time-triggered system and the flexibility of the event-triggered system.

## 1.4 Activities

In this section, we will describe some of the activities performed during the author's time as Ph.D. student. We also present some activities not directly figuring in the main part of the thesis, but which constitute background and related work behind the thesis. Some of this is presented in detail in the appendices.

This doctoral thesis is a continuation of the work presented in the licentiate thesis [Cla99], where we followed a top-down design approach for studying the communication between nodes. Thus, we have studied the requirements and design tradeoffs that affect the communications system. This provide us with a good understanding of the difficulties and requirements of constructing a reliable communication system.

In order to get a better overview of the requirements and difficulties of reliable and cost efficient communication, we have studied how to design dependable distributed systems. As part of this work the XBW-model [CPS98, Cla99] was developed together with the partners within the European Brite-EuRam III project X-By-Wire [X-B98]. This conceptual model facilitates the development of distributed-embedded systems. The model describes the time behavior and distribution properties of a system in such a way that static scheduling and systematic fault tolerance can be applied. The conceptual model also enables the definition of an appropriate fault model. This fault model along with the XBW-model allow efficient and systematic use of well-known software based error detection methods. We also show a short example using this model base on a Steer-By-Wire system. The X-By-Wire model highlights some of the important factors that must be considered when developing an embedded distributed system. The model emphasizes the importance of interfaces between different objects, such that they have a correct interface and prevent faults from disseminating through the system. Specifically, inter-process communication is targeted where the communication can both be within a node or externally using the communication media.

In distributed systems, there is an additional focus on node external task communication as it uses the communication media for such information transfer. The chosen communication architecture will affect the

system design, as it has affect on the transfer rate, response times, reliability, etc. Thus, it is very important how we choose the communication architecture. To study these effects, we have used the computer control of a brake-by-wire system in a heavy-duty truck as a case study. From this case study, relevant requirements have been extracted and we have studied the efficiency using different design approaches, considering the effect on reliability and communication. Naturally, the results from this study are limited to this case. However, many systems have similar requirements and configurations for which this investigation is informative. As this serves as a background to the main result of the thesis, we have appended this report in Appendix A.

As we described in PS1, achieving and maintaining a common time view is essential in a time-triggered system as they fundamentally depend on time-sharing of the communication media. Mechanisms for resynchronization are also necessary after a perturbation has forced a node to restart. Focusing on broadcast-media based communication protocols in TDMA environments, we present a novel, synchronization approach with cost efficiency and reliability as the main drivers. This approach utilizes information about each node's unique message length as node identifiers over the synchronization process. Furthermore, our initial start-up synchronization approach is fault-tolerant and has a bounded start-up time. It avoids start-up collisions by postponing retries after a collision. We also present a resynchronization strategy that incorporates recovering nodes into synchronization.

In order to prepare for the work with PS2 and PS3, we have investigated existing multiple-access methods appropriate for hard real-time systems.

In PS2, we described the differences amongst two contemporary media-access methods, the time-triggered approach and the event-triggered approach. As part of this work, we have compared these two methods to see how they behave under different conditions. The intention is to help system designers determine how these two approaches differ under varied operational scenarios. This has been accomplished by two types of simulations where we have studied the behavior of these approaches. The first simulation type used a system with varied task and communication mix. In the second type, we studied the system by looking exclusively at the communication media. The work does not aim to to establish the basic properties of the basic communication protocols and approaches, as they are well known. Instead, we have focused on quantifying these differences and establishing the specific operational conditions where each technique works best. The intent is to provide the system designer with information about the preferred operational profiles

for each TT or ET schema to be able to conduct quantifiable trade-offs during system design.

We have developed our perspectives from the investigation of existing media access methods, as well as the study of the event-triggered and time-triggered paradigms. On this basis, We have developed efficient approaches to provide for composite event-triggered and time-triggered services within a common time-triggered paradigm such that we can achieve both the predictability and the flexibility inherent in time-triggered and event-triggered systems, respectively.

## 1.5 Main Contributions

We briefly state our main contributions presented in this thesis. Focusing on the composite provisioning of efficient fault-tolerant and real-time requirements, the contribution of this thesis is the development of novel communication approaches via the development and investigation of:

1. A novel synchronization approach for broadcast-media based communication with high reliability and efficiency, which is intended for TDMA environments. Unlike virtually all existing techniques, this method does not require explicit transfer of node id's.

2. Investigation of different multiple-access methods appropriate for hard real-time, followed by:

   - A quantitative assessment of the basic capabilities and limitations of the time-triggered and event-triggered approaches under different working scenarios. This study is intended as a basis for design decisions for communication systems combining issues in schedulability/predictability and efficiency.
   - Proposed new solutions for composite provisioning of the event-triggered and time-triggered paradigms, such that we maintain the flexibility and determinism emanating from the event-triggered and time-triggered approach, respectively.

## 1.6 Disposition

The disposition of this thesis is as follows: Chapter 2 presents our novel synchronization approach for initial startup as well as re-synchronization of a TDMA system. Chapter 3 presents a more in-depth analysis of the behavior of two common media access methods for embedded real-time systems, where we study the efficiency of time triggered communication

compared to event-triggered communication. The final contribution is presented in Chapter 4, describing solutions how to send event-triggered information on a time triggered channel, both based on existing technologies and new approaches. In Appendix A, which follows, we have some background material, which consists of an investigation of communication architectures, and their impact on bandwidth and reliability. It also treats how to map control algorithms to fit with these different communication architectures, based on [CGS00].

# Chapter 2

# Synchronization Approaches in Time-Triggered Environments

IN this chapter we have developed low-cost and fault-tolerant synchronization strategies for Time Division Multiple Access (TDMA) environments, see Section 1.1.1. Specifically, highly efficient, and fault-tolerant communication primitives for safety-critical systems with hard real-time requirements.

Virtually all existing synchronization techniques define and utilize explicit bits for node ID, which are used in the synchronization algorithm. In this work, we remove the explicit bits in messages-frames used for node ID; instead we use each node's unique message lengths as node identifier. The communication primitives are intended for communication protocols where the access method is TDMA. The primary primitives of interest address the start-up behavior of a protocol in a TDMA environment. We investigate how to effectively use the information about messages lengths (ML) as a form of message identifiers upon which a synchronization approach can be built – we term this the ML-approach. A solution for avoiding start up collisions is also discussed using a similar method.

In this chapter we have assumed a broadcast bus as the communication media. Bus systems are prolifically used in many systems today on account of the simplicity and low cost. In computer control, many different protocols utilize a communication bus, for example, CAN [CAN91]. However, with a communication bus a number of different media access strategies exist on how a node can potentially access the bus. The most common way to solve this is by contention resolution, more specifically Carrier Sense Multiple Access (CSMA), where each node senses the bus

for activity and may send when none is detected. This has the drawback that if two nodes send at the same time, the messages will collide. Therefore, it is often combined with Collision Detection where the colliding nodes withdraw if they sense a collision; this is called CSMA/CD and is used in, for example, Ethernet. Although used with great success in Ethernet, the CSMA/CD method is not appropriate for hard real-time systems due to the lack of determinism. With CSMA/CD, it is not deterministically possible to avoid repeated collisions, which effectively constrains estimating worst-case communication times.

To avoid the limitations of CSMA/CD for real-time systems, collisions can be avoided using bit arbitration. With bit-arbitration, messages sent simultaneously are arbitrated using the bit sequence in the beginning of each message. This implies a priority order among messages. Nodes are not allowed to send messages with the same arbitration bit sequence. Using the priority order among messages, a worst-case communication time can be calculated for each message [TB94]. Using bit arbitration, the bus propagation time implies a minimum length in time of the communication bit. Furthermore, bit pulses must be fairly well formed for the arbitration to work. These two facts limit the possible bit communication speed, and it becomes dependent on the cable length.

Token bus (IEEE 802.4) and mini-slotting [ARI95] are other fundamentally different medium access schemes. However, token bus is sensitive to loss of the token and mini-slotting is limited in bandwidth as it is based on the concept of delays. In this chapter, we focus on the TDMA media access method where nodes are pre-assigned time-slots in a repeating schedule. TDMA communication provides a deterministic behavior where, for example, arrival times and worst case delays can be easily calculated. Although TDMA communication has been criticized for its static properties and its consequent lack of flexibility, the properties inherited from the determinism of TDMA communication are very useful, for example, evident timing, composability, easy fault detection, and testing [Kop93].

Furthermore, most computer control systems have real-time requirements where these properties are particularly important, especially when combined with the safety requirements, i.e., hard real-time system, where the consequences are catastrophic if deadlines are missed.

## 2.1 Related Work

Several solutions to TDMA system start-up exist at present, though most deal with systems of limited size where the propagation delay is easily bounded. One possible approach is to use a known bit-pattern at the

beginning of each frame or TDMA cycle [KU95, Sta91]. If a unique bit pattern is used (technique 1), synchronization is obtained when this pattern is detected. If it is not unique (technique 2), the node can search the transmitted bits for maximum correlation with the known pattern and, eventually, obtain synchronization. Both of these cases have delay-related drawbacks for time-critical safety-related applications. Also, if a unique bit-pattern is used, bit-stuffing or similar techniques are necessary to avoid this bit pattern within ordinary messages; in embedded systems for control applications, messages are typically short and in the 100 bits range. Thus, any additional bits results in large overheads. The second technique dictates that maximum correlation must be found, which is complex and adds time overhead. Instead of a bit-pattern, a unique signal level can be used, e.g., a third signal level, other than 0 or 1. However, the extra hardware necessary would probably be more efficiently used to improve the bit encoding. Also, if a faulty node repeatedly issues the resynchronization signal, such a failure would be more severe and difficult to mask than other failure modes that result in invalid transmissions. If, instead, the synchronization information was embedded in a regular message, a message (and a correct checksum) would have to be transmitted successfully in order to achieve synchronization. This is unlikely unless the node is correct.

In the TDMA protocol TTP [KG94], a node is reset and transits to a start-up mode on initialization or when a system-wide communication blackout has occurred. On entering this mode, each node has a unique delay until its first message is transmitted. The unique delay reduces the risk of collisions, but it also means that we cannot continue sending according to the original bus schedule. Instead, the bus clock must be reset when initialization mode is entered. Moreover, if collisions are detected while in this mode, all nodes must reset their clocks [KKH+96]. In the Lightning architecture [DPC+96], the lock-step method is used. During start-up, a node is not allowed to send until its predecessor has transmitted, except for a dedicated "first" node. A time-out is used to detect node failures and allow nodes to start sending even if their predecessor has failed.

The TTP protocol is one of the most well known completely time-triggered approaches. It is a full communication solution, thus, the startup and restart mechanisms we use for comparison is only one part of the TTP protocol. For example, TTP contains services for membership handling and changing operational modes of the communication. However, it should be clear that we, in our approach, only cover startup and resynchronization and have not addressed the membership and operational mode parts. There is nothing preventing this synchronization

approach to be combined with other communication services, and even TTP's approach for operational modes and membership handling.

In the next section we will recall the system model from the introduction, Section 1.1, and include some additional details special for this chapter.

## 2.2 System, Communication, and Fault Models Behind the Approach

**The System Model:** Here we uses the TDMA part of the system model, described in the introduction, and give some more specific details. Recall that, we assume a safety-critical system with real-time requirements and there are $n$ autonomous nodes that communicate via a broadcast bus. The nodes have local counters that are used to control the sending and receiving of messages. Furthermore, the nodes operate in a cyclic manner, where each node sends exactly once, when nothing else is stated, in each TDMA communication round, as shown in Figure 2.1. We also assume, that different nodes may use different message lengths, i.e., a node $i$ sends a message of length $tm_i$.

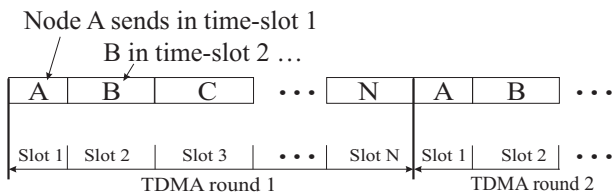


Figure 2.1: A TDMA communication round.

**Communication Model:** We assume two levels of synchronization, (1) synchronization at frame level and (2) synchronization at the communication schedule level. This work assume that an existing approach is used to achieve the frame level synchronization, which ensures that each bit-value is received correctly.

Instead we focus on the communication schedule level, which synchronizes the nodes to the same phase of the communication schedule. That is, to ensure that all nodes agree on the current position of the communication schedule. To achieve this, it is necessary to make nodes exchange time information such that they can agree on a global time. The global time is a coarse-grained time that basically corresponds to the position in the communication schedule.

Each node must store the communication schedule, which will contain

the information of when and what a node should send. The storage requirement for this information is normally relatively small, typically a few kilobits, since we deal with embedded systems with a limited number of nodes, especially when we consider the evolution of today's computers and memory.

The global time-base can be used to synchronize distributed tasks and minimize delay and jitter by relating execution times of tasks to the TDMA rounds, as has been established by [KG94, KKH+96]. This eases the scheduling of periodic tasks.

The intended area for these communication primitives is hard real-time systems that also need to be cost-efficient. The intent is to maintain as low complexity of the primitives as possible. Communication bandwidth for embedded system is sparse and has developed very slowly when compared to the growth in processing efficiency. Thus, it is important to have efficient protocols with limited communication overhead.

**The Fault Model:** The initial synchronization algorithm requires a majority of nodes to synchronize such that different smaller groups do not form disparate cliques. This puts a limit on the number of tolerated faulty nodes to $\lfloor (n-1)/2 \rfloor$, where $n$ is the number of nodes in the system. We assume that nodes follow *fail-silent* semantics, as discussed in the introduction.

For the bus, we assumed omission failure semantics where messages are either received correctly and on time or not at all. Recall that this design, (fail-silent nodes and a bus system) efficiently exclude timing failure on the bus. Similarly, Byzantine failures are not considered as they are avoided by design, using a bus combined with message checksums.

Thus, the initial synchronization will tolerate $\lfloor (n-1)/2 \rfloor$ node failures and message omissions. The number of message omissions may affect the synchronization time, which will be discussed in Section 2.5. A node recovering from a failure will need resynchronization in order to send messages. A node using our ML-approach for synchronization will immediately regain synchronization after the first correct message reception, since it can then determine the sender of the message and thus where in the communication schedule the system is. Thus, assuming normal operation, message omissions will only affect the time for resynchronization of the node.

In the next section we discuss our approach along with current approaches on initial synchronization and resynchronization, highlighting properties and shortcomings, in order to put our approach in perspective.

## 2.3 Initialization and Resynchronizations: Outlining the Basic Approach

The initialization problem of TDMA communication is that nodes can only be synchronized by sending messages on the broadcast bus. However, to be able to send messages on the bus such that collisions are avoided, the nodes must be synchronized in the first place. This leads to a situation where, similar to CSMA/CD, no upper bound can be put on the initialization. For real-time systems and especially hard real-time systems, a bound on the startup time is desirable.

**Current Approaches:** The usual way to exchange time and thus the position in the communication schedule is to send the time of local clocks explicitly in messages. An algorithm can then be used to agree on the global time. We advocate an approach where the message arrivals are clocked and these time values are used for synchronization. This can be done since messages are pre-scheduled and these time values will reflect the local time of the corresponding sender. That is, we can extract the sender's opinion of time, as it sends the message according to its local clock. To succeed with the extraction we must first have a method of identifying the sender of the message. Then the static schedule unambiguously provides the send time. The difference from the scheduled send time and the local time when the message was received, is used to create a correction of the receivers' clock.

A message identifier is usually included in the beginning of messages, as **id**-fields. When using static scheduled messages, the reception time of the message can serve as the **id**, thus making the **id** field of messages unnecessary under normal operation. This assumes that nodes have synchronized their local clocks, i.e., the nodes have agreed on the position in the communication schedule. Thus, before the nodes are synchronized or when nodes lose synchrony, explicit message **id**s are necessary. Thus, nodes need to send explicit message **id**'s until nodes are synchronized.

To handle initial synchronization and resynchronization we can send the message-**id** in all messages, as done in DACAPO [RLST95]. Another way of achieve synchronization is to send special initial messages, as done in TTP [KG94]. Sending the **id** in all messages will add extra overhead, which is only useful at startup and at synchronization. Thus, sending special messages at startup appears as a good idea, but then we increase the complexity by adding an extra communication mode at startup. Furthermore, resynchronization of nodes is not handled, i.e., reintegration of a node that has lost synchronization. In such a case a membership service must be introduced. A third alternative is to let a node send periodic messages with resynchronization information. If the node that

sends such a message fails, nodes that have lost synchronization will not be able to reintegrate. Therefore, additional nodes have to send messages with resynchronization information to tolerate failures. The disadvantage is the requirement of an extra mode and the additional overhead at runtime.

**Proposed Approach:** Our solution takes advantage of information that is inherent in statically scheduled communication. Instead of including the message **id** in each message, we use the length of messages as the **id**. This implies that messages have different lengths and that the lengths of the time-slots correspond to these message lengths. With a static schedule and the length information, a message receiver immediately knows who the sender is. However, this requires that each node has a list of the length of messages and to which node they correspond. The unique message lengths of nodes are chosen according to the amount data each node need to transfer, i.e., the node with most data to transfer (per time unit) will have the largest message. When nodes require the same message length we can break the symmetry by differentiating the message length of these nodes. We can also ensure that there will be unique sequences of message lengths to synchronize on, which we will come back to later in this Chapter and in Section 2.6.3.

This approach is very useful both during start-up and resynchronization, as a node will require only one message receipt to be able to achieve system level synchronization.

Using this approach, nodes will know the position of the communication schedule as soon as one message has been received correctly. However, using different message lengths will not prevent messages from colliding. Making a new retry one TDMA cycle later can lead to a new collision and, in the worst case, lead to an infinite sequence of collisions. The usual work-around for this is to wait a random delay before resending, usually called exponential back-off[1]. This usually works but for hard real-time systems, a bounded time on startup and resynchronization is required.

To achieve a bounded start-up when messages collide, each node will delay its retry by a short but unique time period. We will show that, by using time periods that are short compared to the cycle time, the worst-case start-up time is bounded. This worst case scenario is extremely unlikely and we will therefore show the average startup times of this approach, using simulations.

In most systems, the nodes are likely to have different requirements on

[1]The exponential back-off strategies used in CSMA do not provide a guaranteed bounded time to bus access. Predictable bounded times are essentially required for safety critical applications.

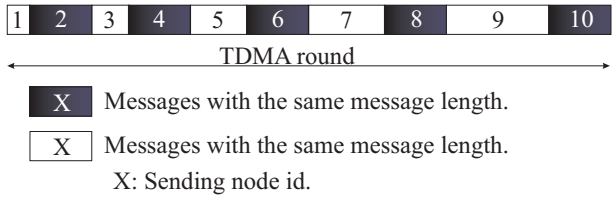


Figure 2.2: A TDMA communication round. Nodes with odd **id** numbers having unique message length. Nodes with even **id** numbers all have the same message length.

bandwidth, thus the requirement of unique message lengths is normally not a limitation. It offers more flexibility than a system using messages of the same length. The requirement of unique message lengths can even be relaxed if some nodes need to send the same amount of data. Nodes can then use messages with the same length by adding simple constraints on sending at the initial synchronization. Such a constraint could, for example, be that only messages longer than previously sent messages are allowed to be sent. This simple algorithm will allow many messages with the same message length. Although it is easy to come up with communication schedules for which the suggested algorithm would not work, there are in most cases other modifications of the algorithms that will work. The simplest solution to solve the problem is however to make a message slightly longer, i.e., add padded bits.

In a situation where we have many short messages requiring similar amount of sent data, it might be difficult to choose different lengths for each node. This situation can easily be handled with two adaptations: (1) at startup, only nodes with uniquely identifiable messages lengths are allowed to send. This means that during startup the received messages must be identifiable, such that nodes can synchronize with them. In Figure 2.2, we show an example where only nodes with odd **id** numbers are allowed to send. (2) During resynchronizations, at least a unique sequence of lengths is sufficient as it can be used to locate the position in the TDMA cycle. For reintegration, it is also sufficient with a unique sequence of message lengths. In Figure 2.2, a resynchronizing node can be synchronized as soon as a message from a node with odd **id** is received. Thus, for startup, only nodes with unique message lengths can send, such that the startup is fast.

### 2.3.1 The Node-Level Synchronization Operations

On this background, we now detail the node synchronization process. Nodes work in three different modes depending on the level of synchronization achieved, see Figure 3. In Mode (1) *normal operation* a node is synchronized and sends according to the preassigned schedule. Mode (2) *resynchronization mode* is entered when a node has lost synchronization or messages from less than half of the nodes are received. Mode (3) is the *recovery mode* that a node enters at startup and after a disturbance preventing message reception for the duration of a full TDMA cycle. The rules are based on those described in [Lön99a]. We first provide some relevant definitions that are used in the synchronization protocol description in Section 2.3.1.

#### Definitions

**n** is the number of nodes in the system.

**$\mathbf{tm_i}$** The time it takes for a node $i$ to send its message $M_i$, i.e., this time is proportional to the message length.

**$\mathbf{INC_i}$** Each node $i$ is alloted a unique time period $INC_i$ used to delay the nodes transmission after a collision.

**$\mathbf{tm_{max}}$** The time it takes for the node with the longest message $M_{max}$ to send its message, i.e., the time it takes to send the longest message.

**$\mathbf{TC_i}$** The time counter for node $i$, keeps track of the current time in the schedule. With this pointer each node will know when to send.

**$\mathbf{ST_i}$** The send time of node $i$.

**MRvec** Message Receive Vector, containing information on whether the latest $n$ messages were received correctly or not. In a system with eight node's ($n = 8$) where a node's $MRvec$ contains three 0's and five 1's, {0,1,0,0,1,1,1,1} means that this node has received five messages correctly and three incorrectly or not at all. $MRvec$ is a FIFO list where a new message is inserted and the oldest are shifted out. In case of an incorrect message or no message is received a 0's is inserted into the vector, otherwise, if a correct message is received a 1 is put in the vector. We denote the number of 1's in $MRvec$ with $ones(MRvec)$.

**$\mathbf{SC_i}$** The silence counter for node $i$, timing the period from last bus event or disturbance. The silence counter is reset every time any messages/traffic is sensed on the bus.

Having introduced the definitions, we now present the synchronization protocol, with the operational modes and their corresponding operation.

### Synchronization protocol, modes and operation

Each of the following operations is performed at each node. A node will also treat message receptions from itself in the same way as other nodes, e.g., updating the $MRvec$.

**Normal mode:** When nodes are synchronized and messages are received from more than half of the system nodes.

1. For each correctly received message $M_s$ sent by node $s$, the receiving node $r$ updates the time counter $TC_r$ by adding the time $tm_s$ corresponding to the received message $M_s$. Thus $TC_r = TC_r + tm_s$. The received message vector $MRvec$ is updated and a 1 is inserted into the vector.
2. If an incorrect or no message was received by a node $r$, the time counter $TC_r$ of node $r$ is incremented with the duration of message $i$, $tm_i$ of the expected message. The received message vector $MRvec$ is updated and a 0 is inserted into the vector.
3. Enter Resynchronization mode if messages from less than half of the nodes have been received correctly, i.e., the number of ones in $MRvec$ is less than half of the number of nodes, $ones(MRvec) \leq \lfloor n/2 \rfloor$.
4. A node $i$ will send its message when the time counter is equal to the send time, $TC_i = TS_i$.

**Resynchronization mode:** Having sent a message in recovery mode, a node enters resynchronization mode and waits for reception of messages from half of the nodes. (Note: No message transmission is made in Resynchronization mode).

1. When a correct message $M_s$ is received by node $r$, set the time counter $TC_r$ according to the end time of this message, $TC_r = ST_s + tm_s$. The received message vector $MRvec$ is updated and a 1 is inserted into the vector.
2. If an incorrect or no message was received by a node $r$, the time counter $TC_r$ of node $r$ is incremented with the sending time $tm_i$ of the expected message. The received message vector $MRvec$ is updated and a 0 is inserted into the vector.
3. Enter Normal mode when messages from a majority of nodes have been received correctly.
$$ones(MRvec) > \lfloor n/2 \rfloor \tag{2.1}$$
4. If the bus has been completely silent for one communication cycle (silence counter is greater then the time period, $SC >= T$) enter recovery mode.

**Recovery mode:** The start mode, where nodes enter at initial-startup. A Node starts by listening to the bus for one communication cycle; if no message was received it will send its message. Nodes also return to this mode in case of complete loss of synchronization.

1. A node $i$ is allowed to send a message if the following condition is fulfilled: The local time counter $TC_i$ indicates that it is node $i$'s turn to send, we refer to this time as $ST_i$.
2. When a correct message $M_s$ is received by node $r$, set the time counter $TC_r$ according to the end time of this message, $TC_r = ST_s + t_{ms}$. The received message vector $MRvec$ is updated and a 1 is inserted into the vector.
3. If an incorrect or no message was received by a node $r$, the time counter $TC_r$ of node $r$ is incremented with the duration $tm_i$ of the expected message. The received message vector $MRvec$ is updated and a 0 is inserted into the vector.
4. If a collision was detected by node $i$ when sending its message, it will immediately stop sending and postpone its retry with $INC_i$ time units, i.e., next transmit time is $INC_i + T$, where $T$ is the period time.
5. When one message has been successfully sent, enter Resynchronization mode.
6. Enter Normal mode when messages from a majority of the nodes have been received correctly in $MRvec$, i.e., $(ones(MRvec > \lfloor n/2 \rfloor)$.

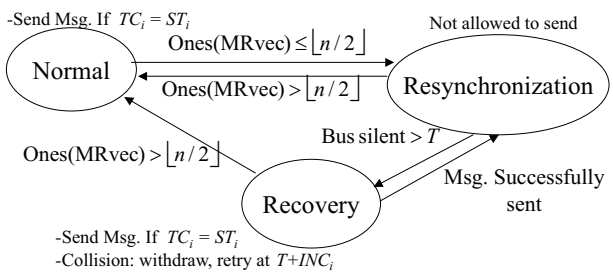


Figure 2.3: Communication controller states

To reach normal mode (Fig. 2.3), we require message receipt from a majority of the nodes, thus we tolerate $\lfloor (n-1)/2 \rfloor$ faulty nodes. The effect of failure during start-up and resynchronization is a delay in the startup time.

It is important to avoid repeated collisions of messages which would prevent the system from synchronizing. This is handled in the recovery

mode above. Each node $i$ is assigned a time increment $INC_i$, significantly shorter than the period time T. If a collision occurs, colliding nodes will postpone their next retry with a time equal to the increment $INC_i$. Thus, the next send time $t_{k+1}$ for node $i$ will be one period plus the time increment, i.e., $t_{k+1} = t_k + T + INC_i$. This will postpone the resending of the message with the time period $INC_i$.

If further collisions occur, a node will keep adding its unique time period $INC_i$ to its time counter. After a deterministic time interval as detailed in Section 2.4, Equation (2.3), at least one message will not collide. All nodes will receive this message and synchronize to it, which will prevent further collisions. Thereby, the initial synchronization completes.

In the worst case scenario the number of collisions will be $\lfloor n/2 \rfloor$, where $n$ is the number of nodes in the system. In Section 2.4 we will derive this figure and show its correctness.

## 2.4 Upper Bound on Startup

In this section we establish the upper bound on subsequent collisions, which is $\lfloor n/2 \rfloor$. In an initial start-up scenario, a node will start in the Recovery mode. In this mode, a node will synchronize with the first received message. Thus, we need to show that the nodes, in a bounded time, will receive an uncorrupted message to synchronize with.

To prove this upper bound on the initialization we use the following assumption:

$$INC_1 < INC_2 < \cdots < INC_n << T \tag{2.2}$$

where $n$ is the index of the last node. Thus, node $n$ will use the longest delay after a collision. The minimum time unit must be at least one propagation delay, $\tau$, of the media. Thus, the shortest $INC$ must be at least $\tau$, and the following must differ with at least $\tau$.

The goal is to prevent that an infinite sequence of collisions occur on the bus. This is done by postponing a node $i$'s next send retry with $INC_i$ after a collision, preventing the same nodes from colliding at their next retries. Using these assumptions, we can draw the following conclusions:

1. The worst case number of collisions occurs when all nodes collide but only in pairs. That is, the first two nodes collide then the next two etc. In this worst case scenario, all nodes will collide again at the next retry, but in different pairs.

2. For such a worst case scenario to occur, nodes with the longest $INC$ delay must collide first followed by collisions of pair of nodes with

decreasing $INC$ delays, i.e., node $N_n$, with $INC_n$, will collide with node $N_{n-1}$, with $INC_{n-1}$, and then node $N_{n-2}$ with $N_{n-3}$ and so on. If the two nodes with the shortest delays ($INC_1$ and $INC_2$) collide first they will not collide again. Furthermore, they will not collide with any other nodes as other nodes delay their retries even further. Thus the collisions would end at their next retry.

For each collision, the "fast" node will move forward a step and must collide with a node that has a lower $INC$ delay. To explain the idea we give a short example with 10 nodes where these possible collisions give a worst case scenario. In each row, ($N_i$,$N_j$) indicates that nodes $N_i$ and $N_j$ have collided and $K$ indicates the collision number:

$K = 1$ $(N_{10}, N_9)(N_8, N_7)(N_6, N_5)(N_4, N_3)(N_2, N_1)$

$K = 2$ $(N_9, N_7)(N_{10}, N_5)(N_8, N_3)(N_6, N_1)(N_4, N_2)$

$K = 3$ $(N_7, N_5)(N_9, N_3)(N_{10}, N_1)(N_8, N_2)(N_6, N_4)$

$K = 4$ $(N_5, N_3)(N_7, N_1)(N_9, N_2)(N_{10}, N_4)(N_8, N_6)$

$K = 5$ $(N_3, N_1)(N_5, N_2)(N_7, N_4)(N_9, N_6)(N_{10}, N_8)$

Based on this idea of postponing the retries with a unique delay, we prevent infinite sequences of collisions. Thus, for $n$ nodes, we will get a maximum of $\lfloor n/2 \rfloor$ collisions. An odd number of nodes will only require that we have a number of triple collisions to get a worst case, and this reduces the probability of collisions.

The longest delay before a message is sent without collision is then:

$$t_{startup} = (\lfloor n/2 \rfloor + 1)(T + INC_n) \tag{2.3}$$

where T is the TDMA round time and $INC_n$ the largest increment among the nodes.

## 2.5 Properties and Overhead

In this section, we discuss and show the properties of this synchronization approach. The overhead of synchronization can be manifested in three ways, namely (1) communication overhead, (2) synchronization time overhead and (3) computation/memory overhead. The communication overhead relates to additional data that must be transferred in order to achieve synchronization. The time overhead relates to the additional timeouts, retransmissions etc. Finally the computation/memory overhead is related to processing and memory storage which is required in order to accomplish resynchronization.

This synchronization approach is efficient in the sense that it has low overhead. However, there is invariably some overhead related to communication services like synchronization. In safety-critical real-time systems, the synchronization overhead and communication overhead are normally most critical. Computation time and memory is relatively cheap in comparison. Therefore, we favor a solution that may have a larger computation overhead but smaller time and communication overhead. The following list summarizes the main overhead contributors in this approach:

**Communication:** No additional messages are sent and no extra information bits are needed to achieve synchronization, except in the rare cases when we need to add extra bits to get unique message lengths.

**Time:** Synchronization is accomplished as soon as a message is received. It may be delayed by collisions and faulty nodes, bounded as given in Equation (2.3), Section 2.4.

**Computation/Memory:**

- Each node carries a list of "$n$", i.e., one for each node in the system, entries containing all message lengths and the corresponding nodes, it also include the knowledge about the time slot it should be sent out in.
- A counter which keeps track of local time, i.e., the point of the communication schedule the system has currently reached.
- A vector with the $n$ last received/expected messages, with only one bit necessary for each node, indicating received or not received messages.

To analyze the proposed Message Length (ML) approach, we compare it with two existing approaches for initial synchronization and resynchronization. The approach used in DACAPO [RLST95] (*D-approach*) uses fixed length messages where one or two special nodes change its send time in order to resolve collisions. The second approach is based on the one used in TTP [KG94, TTP99], the (*T-approach*), see also Section 2.6.3 where nodes send special start frames and each node uses dedicated back-off times. Finally, we include an approach that behaves as ML except that it uses exponential back-off instead of time increments to resolve collisions. This approach will be referred to as $ML_{exp}$. The exponential back-off uses a random delay before retry, and additional collisions exponentially increase the time range from which the random delay is chosen. The exponential back-off is used in, for example, the Ethernet. This method would require a random number generator in each node instead of our increment value calculated and stored pre-runtime. Although slightly more complex in implementation, $ML_{exp}$ is independent of system size.

In Table 2.1, the main differences in overhead for these start-up scenarios are summarized. The main categories we have compared are (a) communication bandwidth overhead (in number of bits), (b) storage overhead, (c) bounded/unbounded startup time and (d) maximum time for a node to resynchronize. In our $ML$ approach and $ML_{exp}$, there are no communication overheads, since information is transferred using message lengths. The required storage is proportional to the number of nodes $n$ such that a message can be identified by the message length. The major

difference between these two is that $ML$ has a bounded startup time, see Section 2.4.

For the resynchronization approach, the time to resynchronize a recovered node is totally dependent on the time of reception of a first message. The time to send messages from node $i$, i.e., the duration that message occupies the bus, is $tm_i$ and we assume that we index the nodes such that the send duration for node $i$ is shorter than for node $i+1$, i.e., $tm_1 < tm_2 < \cdots < tm_i < tm_{i+1} < \cdots < tm_n$. Thus, the worst case resynchronization time, i.e., the longest time it can take for a node to receive a message, would be:

$$t_{resync} = tm_{n-1} + tm_n \tag{2.4}$$

In case of $f$ faults, i.e., message omissions or node crashes, we have:

$$t_{resync} = tm_{n-1-f} + \cdots + tm_{n-1} + tm_n \tag{2.5}$$

As we see in Equation (2.5), $t_{resync}$ will increase for each extra fault. As a pessimistic approximation, we can write $(f+1) \cdot tm_{max}$ if $f$ is the number of tolerated faults and $tm_{max}$ is the longest message length.

The TTP approach requires special initialization messages to be sent that include information about the sender. A node is synchronized when it has received an initialization message; this is equally true for a resynchronizing node. Thus, initialization messages must be sent during normal operation to allow a recovering node to resynchronize. Furthermore, to tolerate failure of the initialization message sender, $(f+1)$ nodes must send this types of message. The number of bits required in initialization messages is $(f+1)log_2(n)$, where $log_2(n)$ bits are needed to identify the sender and such a message must be sent $(f+1)$ times to tolerate $f$ failures.

| Method | Comm.(bits) | Memory (bits) | Startup | Resynch. (secs) |
|---|---|---|---|---|
| Message Length ($ML$) | 0 | $\propto n$ | B | $(f+1) \cdot tm_{max}$ |
| D-Approach | $log_2(n) \cdot n$ | $log_2(n)$ | B | $(f+1) \cdot tm$ |
| T-Approach | $log_2(n) \cdot (f+1)$ | $log_2(n)$ | B | $f \cdot t_d + tm_{im}$ |
| $ML_{exp.}$ back-off | 0 | $\propto n$ | U | $(f+1) \cdot tm$ |

Table 2.1: Properties of startup/resynchronization approaches. B=Bounded and U=Unbounded.

The start-up time for this approach is normally bounded, since all nodes should listen to the bus and reset their local clock at this time. After such a reset, nodes will wait for a node-specific time, based on the

requirement, to send their message. For this to work and to be sure of avoiding additional collisions, all nodes must have sensed the collision. By increasing the time between two successive initialization messages, reduce the overhead stemming from initialization messages. This will however increase the worst-case time for resynchronization of a recovering node. The worst-case resynchronization time depends on the time between and length of initialization messages, $f \cdot t_d + tm_{im}$, where $t_d$ is the largest time between initialization messages and $tm_{im}$ is the length of those messages.

In the D-approach, the *sender ID* is always included in the message and will result in a communication overhead of $nlog_2(n)$ per TDMA cycle. There is no extra storage needed for either the *D-approach* or *T-approach* for the startup. This should not be confused with the fact that TTP-controllers already store total information about the communication schedule, including messages lengths. This extra overhead is used for other purposes than the startup synchronization and resynchronization.

Table 2.1 shows that the $ML$-approach combines a bounded start-up time with low communication overhead and fast resynchronization.

## 2.6 Simulations

To show the normal start-up behavior of our approach, we have simulated the initial startup synchronization and measured the time for all nodes to reach the Normal mode. This has been done in a number of scenarios, such as under normal fault free condition as well as with faulty nodes during the startup.

We have assumed that we have a bus system where the bus is no longer than 40 m. The propagation delay for a 40 m cable is approximately 0.2 $\mu$s, and we have chosen our communication bits to twice this time. This affects the startup times but as we have used the same for all simulated protocols it does not affect the relative comparison between protocols. In these simulations, the basic time-units are 0.4 $\mu$s, i.e., the basic bit transmission time.

The message lengths are unique and chosen as a multiple of the basic time unit, of 0.4 $\mu$s. For the time increments ($INC_i$) it is important that they are longer than the propagation time. Furthermore, they should differ by more than one propagation time-unit each, such that after a collision between two nodes, these nodes will not collide again. In our simulations, $INC_i$ is chosen starting with one time unit, i.e., two times the propagation time, and for each additional node we add two time units, e.g., $INC_i = 1, 3, \ldots, (1 + 2 \cdot i)$ for $i = 1$ to $n$ where $n$ is the number of nodes.

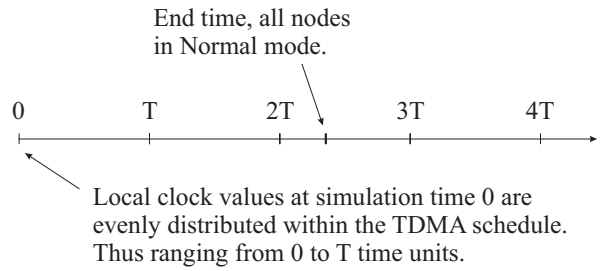


Figure 2.4: Startup initialization.

The start times of the local clocks, i.e., the time counter $TC$, have been evenly distributed in the time interval (0,T) where $T$ is the TDMA round time, see Figure 2.4.

### 2.6.1 Normal Operation

We have started our simulations by finding the startup times under conditions when all nodes work correctly. The message lengths have been chosen randomly from 24 bits and up, such that the Cycle Time (CT) of a communication round is equal to a certain chosen period. The selected periods are between 0.2 and 1.6 ms. The size of the system varies between 6–24 nodes.

For each CT (cycle time), 50 sets of messages with randomly generated lengths were used. We have initially chosen to randomly generate messages between 24 bits and $CT/(n/2)$, where $n$ is the number of nodes. The 24 bits is not a limit but has been chosen as it is reasonable to assume that smaller messages than that are seldom used. However, if messages of equal length were generated they were separated by decreasing the length of one of them. Therefore, a few messages may have been separated such that their length are below the 24 bits.

The sum of all $n$ messages $S_r$ should be CT, therefore each randomly generated message length $mlr_i$ was adapted according to Equation (2.6). The used message length $ml_i$ is then:

$$ml_i = (mlr_i - 24) \cdot \frac{S_r - 24 \cdot n}{CT - 24 \cdot n} + 24 \tag{2.6}$$

This ensures that messages are randomly generated starting from 24 bits and the total sum is $CT$. The message length distributions for the 6-node and 24-node system are shown in Figure 2.5 and Figure 2.6.

We use the 24-node case to show the operational capability of our approach for a relatively large system size as well. The system size of 6

and 24 nodes will also be used for the simulation of the fault scenarios.

For each of these 50 message sets 1 000 startups were run, such that in total 50 000 startups were run for each CT-case. In Figures 2.7–2.10 we see the average, maximum and minimum startup times for these simulation runs.

As seen in Figures 2.7 – 2.14 the startup time increases linearly with increasing Cycle Time (CT). This means that the startup time increases when the average message length increases while using the same system size. This also applies to the maximum and minimum startup times.

The average startup time is dependent on the CT. The startup time is basically a fixed factor of the CT. In the 6-node case the startup time is close to 1.8 times the CT and slightly less for the 24 node case where it is 1.6 times the CT.

### 2.6.2 Fault Scenarios

In this section we will see how a system startup behaves when one or more nodes fail during startup. A number of failure scenarios can occur during the startup of the system.

The main failure scenario we have to consider is when a node falls silent before or during the startup. Since this will affect the startup behavior we have simulated when faulty nodes are silent during the startup. The main effect on the startup is that the average startup times for the working nodes increase, as can be seen in Table 2.2. It also shows the relatively small standard deviation on the average startup for the three cases and the increasing minimum and maximum startup times, due to faulty nodes, during the simulations. To further show how the startup is affected, we show in Figure 2.15 the relative frequency of the startup times for the different simulation runs.

| Startup time (ms) | No. of faulty nodes | | |
|---|---|---|---|
| | 0 | 1 | 2 |
| Mean | 0,73 | 0,78 | 0,86 |
| Std. Dev. | 0,067 | 0,076 | 0,083 |
| Min | 0,40 | 0,58 | 0,57 |
| Max | 1,07 | 1,18 | 1,21 |

Table 2.2: Simulation result using a 6-node system with 0, 1, and 2 nodes faulty during the startup, the cycle time is 0.2 ms

In order to show how the failure behavior changes for larger system sizes we show in Figure 2.16 the relative frequency of the startup times for a 24-node system. The maximum startup time also increases in the 24-node case and in our simulations the maximum startup time increased

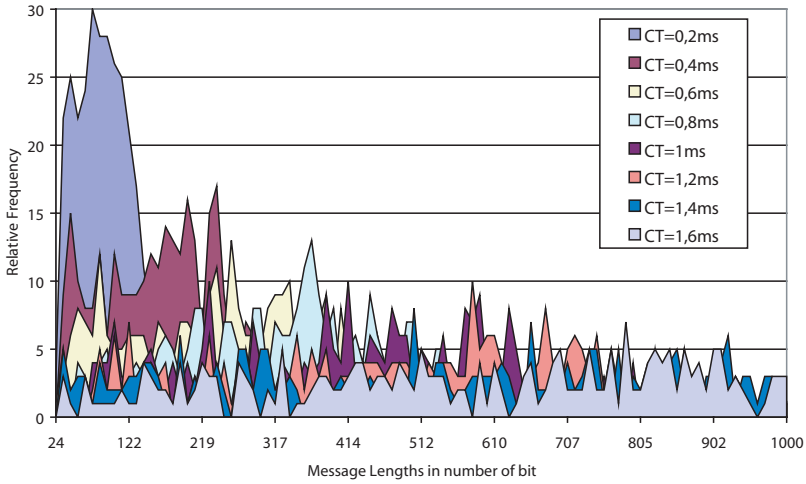


Figure 2.5: Message length distribution with the ML approach for a 6-node system

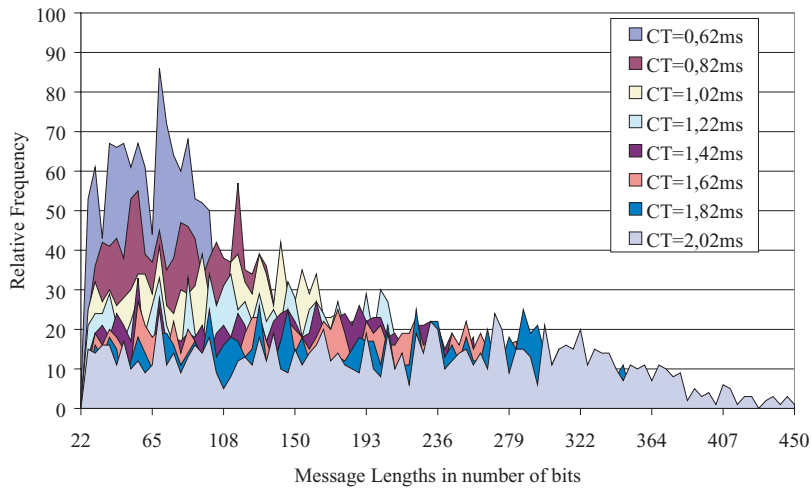


Figure 2.6: Message length distribution with the ML approach for a 24-node system

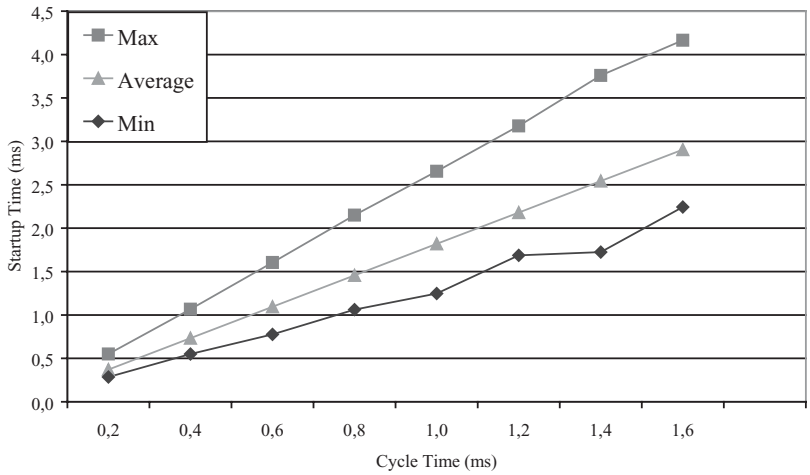


Figure 2.7: Average, Max and Min startup time of 6-node system.

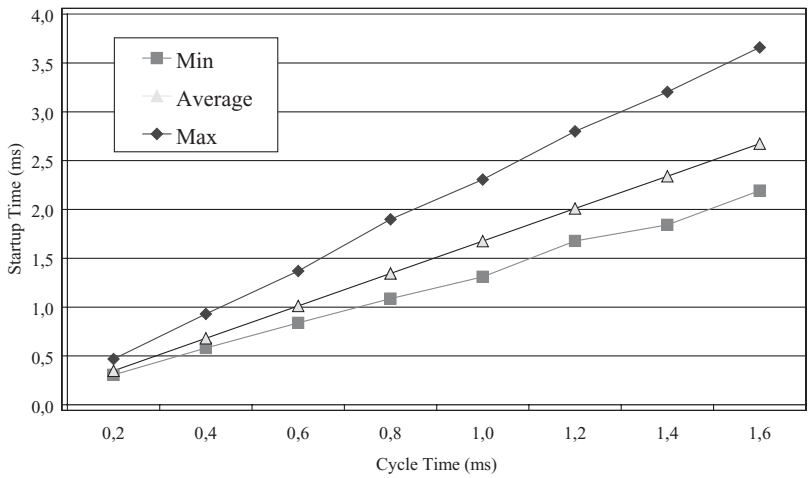


Figure 2.8: Average, Max and Min startup time of 12-node system.

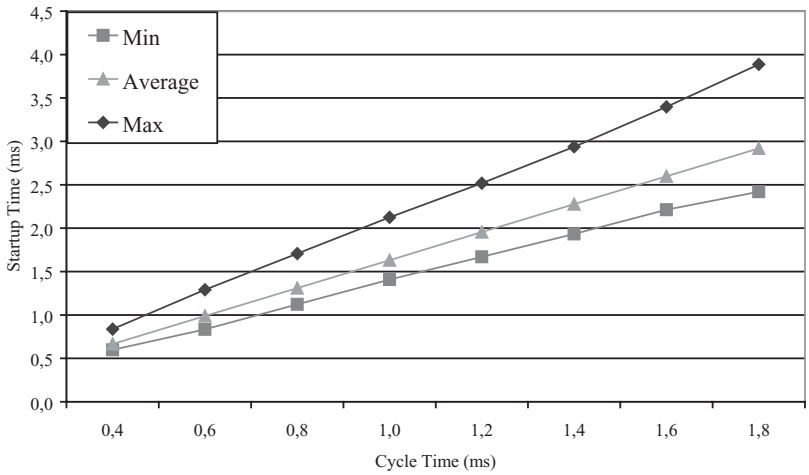


Figure 2.9: Average, Max and Min startup time of 18-node system.

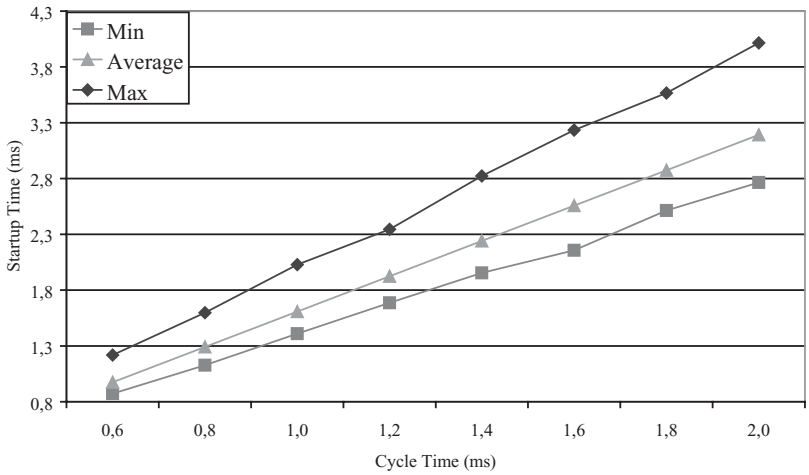


Figure 2.10: Average, Max and Min startup time of 24-node system.

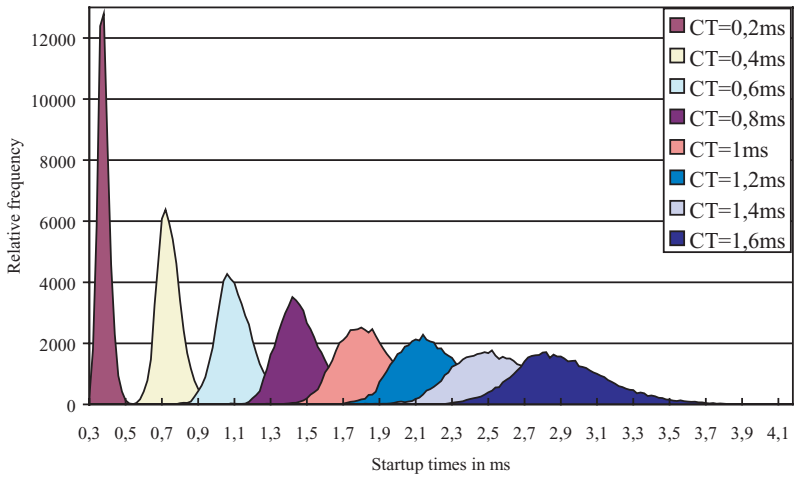


Figure 2.11: The relative frequency of the startup times, in a 6-node system. Each curve corresponds to a Cycle Time, where the leftmost curve has the shortest Cycle Time.

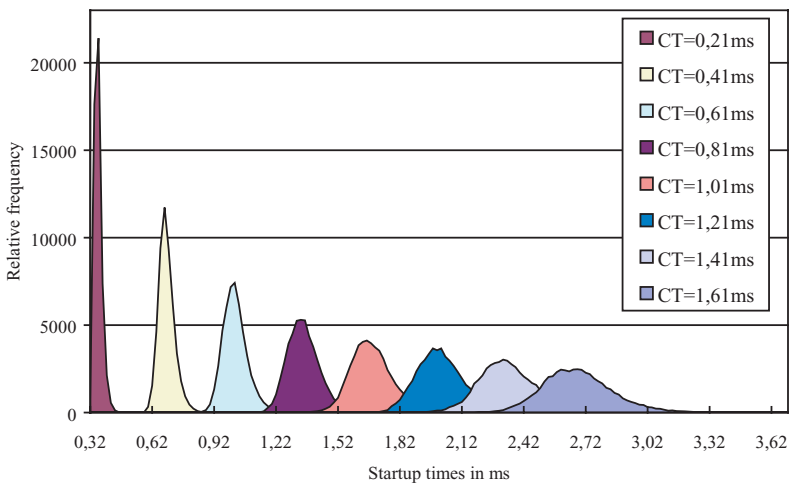


Figure 2.12: The relative frequency of the startup times, in a 12-node system. Each curve corresponds to a Cycle Time, where the leftmost curve has the shortest Cycle Time.

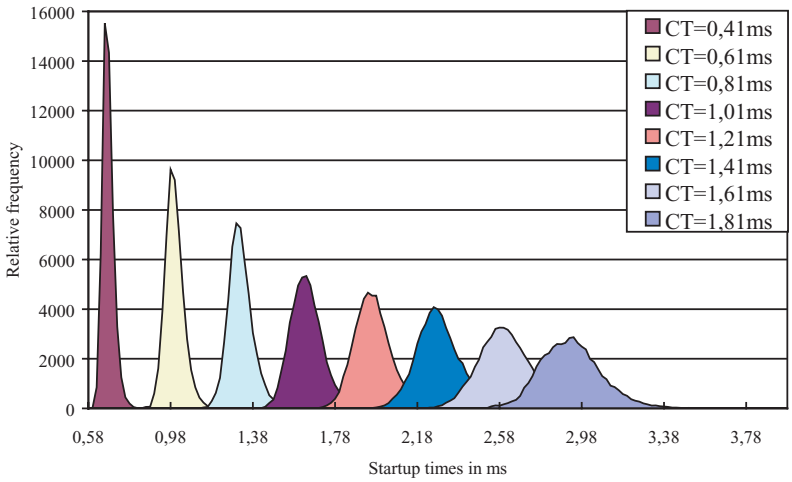


Figure 2.13: The relative frequency of the startup times, in a 18-node system. Each curve corresponds to a Cycle Time, where the leftmost curve has the shortest Cycle Time.

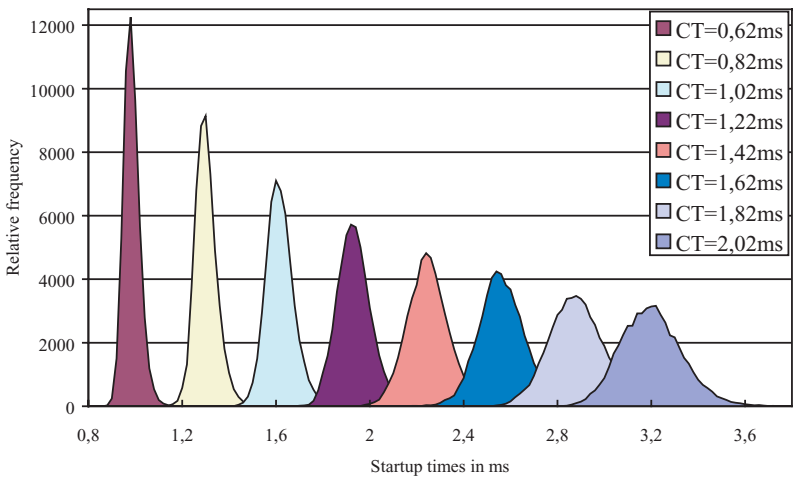


Figure 2.14: The relative frequency of the startup times, in a 24-node system. Each curve corresponds to a Cycle Time, where the leftmost curve has the shortest Cycle Time.

from 1.60 $ms$, for the case with zero faulty nodes, to 1.87 $ms$ for the case with seven faulty nodes.

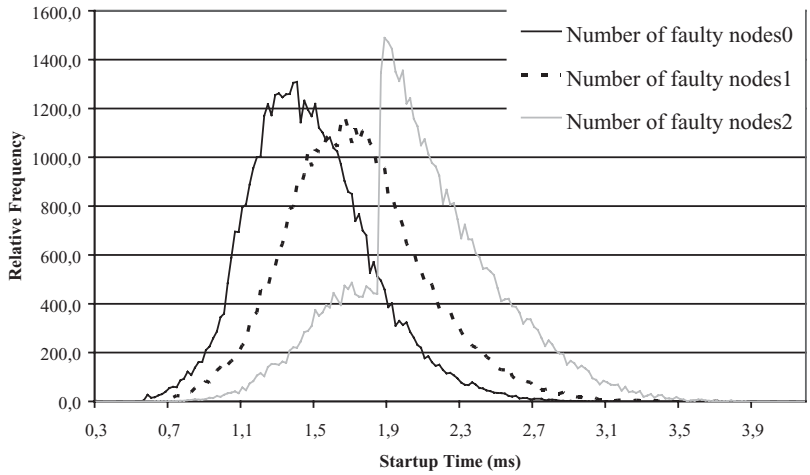


Figure 2.15: The relative frequency of the startup times, in a 6-node system. The cycle time is 0.2 ms. The curves correspond to startup cases were the number of faulty nodes are 0, 1, and 2.

### 2.6.3 Identical Message Length Scenario

An argument against this method is that with larger system size, i.e., with increasing number of nodes and messages, it is difficult to assign unique message lengths. Data transferred in most system are also often multiples of bytes which makes it even more likely that different nodes requires the same message lengths. However, this works with unique sequences of message lengths as well as unique messages. There exist methods of getting unique message lengths and ensuring unique message patterns.

In real-time systems, data exchange often consists of control values, e.g., actuator set-points etc., such data typically needs around 8 to 32 bits per message. These values are packed in messages and transferred on the communication media. Normally, a number of these data values are combined into a message, which controls the message lengths. Each data could however be sent separately, but that would obstruct our main objective, to decrease the overhead, as each message impose an overhead, e.g., for checksums etc. Thus, better message response may come at the expense of decreasing bandwidth efficiency.

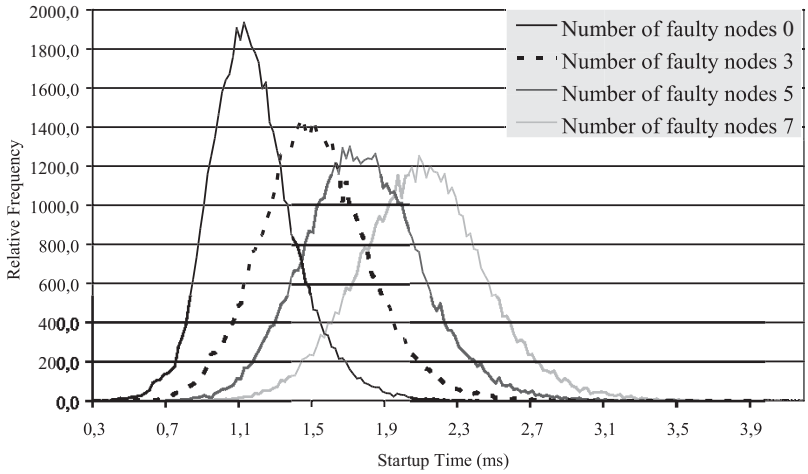


Figure 2.16: The relative frequency of the startup times in a 24-node system. The cycle-time is 0.82 ms. The curves corresponds to a startup cases were the number of fault nodes are varied between 0 and 7.

As larger systems are considered, which increase the number of messages, we may be forced to use messages with equal length. However, a unique sequence of messages can also be used to decide the systems current position of the communication schedule. When using a sequence, a time penalty follows as more than one received message may be necessary to resolve the position in the schedule. We will in Section 2.6.3 show how this time can be reduced under certain conditions.

The sequences of message lengths we need to avoid are identical sequences that are repeated, which makes it impossible to distinguish the exact position of the schedule. Assume that a number of nodes send messages with length x, and another set of nodes sends messages with length y and similarly there are nodes using message length z. Thus x, y and z are non-unique message lengths, i.e., they do not provide unambiguous information of the position in the schedule. In Figure 2.17, we show a few examples on unique and non-unique sequences. The difference among unique sequences is the worst case time to unambiguously decide the position in the communication schedule.

Note that we must use at least one more node with non-equal message length than the number of faulty nodes that are to be tolerated.

Unique sequences:

```
xyyzzzxz:xyyzzzxz:xyyzzzxz:xyyzzzxz
xxxxxxxy:xxxxxxxy:xxxxxxxy:xxxxxxxy
```

Not unique sequences:

```
xyzxxyzx:xyzxxyzx:xyzxxyzx:xyzxxyzx
etc
```

Figure 2.17: Unique and not unique sequences of message received from a bus. Each TDMA-cycle is 8 messages and they are separated with semicolon (:).

### Optimizations

In this section we will describe how we can minimize delays emanating from messages with same length. A sender with a non-unique message length, say node $b$, will be pre-selected if a node receives a message with the length of node $b$, i.e., nodes will assume $b$ was the sender independent who actually sent it. This pre-selected node, $b$, will be determined before runtime. When receiving a non-unique message all nodes will assume $b$ as the sender and thereby the system nodes can be synchronized using any of the messages with that length. For example, if we have a 6-node system with nodes labeled "$a$" through "$f$", $a, b, c, d, e, f$, sending in alphabetical order. We assume nodes $a$ and $b$ to have the same message length. Then, if the other nodes receive a message from node $a$ or $b$ in the startup phase, they will always assume that $b$ was the sender, regardless of the actual sender. All nodes will synchronize to this message and they assume $c$ is the next message to be received.

The drawback of this method is that the message content cannot be used when the first received message is not unique, as the sender is uncertain. This is not a problem during an initial startup or system resynchronization as nodes are synchronized after the first message is received, thus the contents of following messages can be used. We argue that this is reasonable since the primary issue is to get nodes synchronized and it is only one message where the data contents cannot be used, due to an unknown sender. Thus, the only negative effect of this in the initial startup is that the first message cannot be used.

This optimized approach cannot be used for resynchronization of a recovering node, i.e., only in recover mode. A resynchronizing node will have to wait until a unique pattern/message has been received in order to be sure of the current position in the communication schedule.

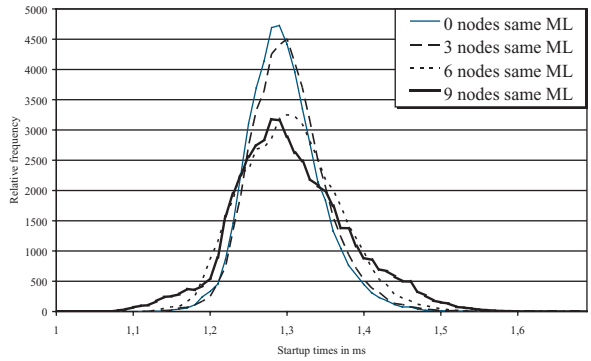


Figure 2.18: The relative frequency of the startup times in a 24-node system using 3, 6 and 9 messages with the same length respectively. The Cycle Time is 0,82 ms

With this optimized method there will be a limited effect on the average startup times. To give an example of this, we have run simulations with a 24-node system and a relatively small cycle time, see Figure 2.18. We have used this as an example, since messages of the same size are more likely needed when using a system with many nodes and relatively short message, such that varying the message lengths is hard. All message lengths have been chosen randomly as described in Section 2.6.1, one message length is then used for a number of nodes. In our simulations we have used the 24-node system with 3, 6 and 9 messages with equal length. The simulation results are shown in Figure 2.18. As the figure shows, there is a limited effect of using the same message lengths when we use this optimized approach. The main difference when the number of messages with equal lengths increases, there is a slightly larger dispersal between the startup times. For comparison we have also included the case with no identical messages.

Without optimization, we have assumed that the receiving nodes will ignore a message with a non-unique message length. This will make the system behave as if nodes with equal messages length have failed, i.e., they are silent. However, in this case they will still send messages and thus increase the probability for collisions. We have not done any simulations for this case since it is very similar to simulations in Section 2.6.2.

We have shown that we get a small impact on the startup times using the optimized method. However, there is still a question of the applicability of this method in reality. How often do we get schedules where we cannot use our method at all, i.e., without using some manual interven-

tion like adding bits to get unique messages. It is hard to find typical real-time application data, for example, industry can seldom provide typical real-time system schedules or information about the amount of data they send. In order to get an indication of how often schedules occur that prohibit our approach, we have randomly generated a large number of schedules. We have then studied this data to see how often our approach can be used and how the maximum delays are imposed due to re-occurring message sequences.

To handle data values that are multiples of bytes, many communication protocols use messages lengths with multiples of bytes as well. We use the same principle and use only messages with multiples of bytes. This naturally reduces the number of possible message lengths in the system. Messages in real-time systems are approximately 24 to 240 bits long excluding overhead, this gives us 28 different message lengths. However, to stay on the pessimistic side with our figures, we will assume message lengths between 56 and 200 bits, which gives us only 19 different message lengths to choose from.

We have generated TDMA schedules, consisting of randomly generated messages that are evenly distributed among these 19 message lengths. These simulations have been done for different system sizes, i.e., with 10, 15, 20, 25, 30, 35 and 40 messages per TDMA-round. We have measured the longest non-unique sequences of messages in each simulation and studied the effect of increasing number of messages. The result is shown in Table 2.3, where we generated 100 000 schedules per system size. The columns show the occurrences of the longest sequence of messages that must be received before the sender's ID can be established. For each system size the occurrences of all the generated schedules are shown, such that the sum in each row is 100 000. In column 1, we see the occurrences where only one message must be received before the senders ID can be established. In column 2, the longest sequence that exists in a schedule before the sender's ID can be established is 2, etc. For each system size we show in the Table the distribution of the longest non-unique sequence. We found that even in quite difficult circumstances, all generated schedules could use our approach. However, with increasing number of nodes it becomes more frequent with repeated sequences of message lengths.

### 2.6.4 Comparison

In this chapter we will compare the simulations of our startup approach with the popular TDMA communication approach TTP/C [KG94, TTP99]. We will also compare with one of our earlier approaches, used

| | Longest non-unique sequence, in # of messages. | | | | | | |
|---|---|---|---|---|---|---|---|
| **System size** | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
| 10 nodes | 55532 | 41742 | 2585 | 141 | 0 | 0 | 0 |
| 15 nodes | 5233 | 82935 | 11177 | 624 | 30 | 1 | 0 |
| 20 nodes | 21 | 74573 | 23872 | 1463 | 66 | 4 | 1 |
| 25 nodes | 0 | 58843 | 38501 | 2511 | 132 | 13 | 0 |
| 30 nodes | 0 | 42861 | 52780 | 4127 | 223 | 9 | 0 |
| 35 nodes | 0 | 28957 | 64719 | 5982 | 330 | 10 | 2 |
| 40 nodes | 0 | 18239 | 73271 | 7963 | 508 | 19 | 0 |
| 45 nodes | 0 | 10669 | 78410 | 10320 | 573 | 27 | 1 |

Table 2.3: Distribution of non-unique sequences divided by the different systems sizes. The sequences are measured in the number of nodes required before they can be resolved. For each system size 100 000 schedules has been measured.

in the DACAPO [RLST95] system.

TTP/C is a time-triggered protocol for distributed real-time systems. It focuses on safety-critical systems and is designed to tolerate faults.

To get nodes synchronized, TTP/C sends special messages with information about the time and the other C-state information. These messages are called *Initialization frames* (I-frames) and are sent at startup and regularly under normal operation such that nodes can resynchronize after a transient failure.

The basic TTP/C startup behavior can under normal conditions be described in three steps as follows [TTP99]:

1. When the nodes are turned on they enter an *Init* mode. In the *Init* mode the nodes run the initialization code.
2. After a node has initialized itself, it enters the *Listen State* where it starts a *listen-timeout* and waits for an initialization message, i.e., the I-frame. If a node receives an I-frame before the timeout it can synchronize itself to the sending node. After the reception of the I-frame a node transfers to the *Active State*, via the *Ready State*. In the *Active State* nodes send normal messages, i.e., messages with data information.
3. If a node does not receive the I-frame before the listen-timeout ends, it will send its own I-frame. After sending this I-frame message the node will wait until it receives a new I-frame message with the same C-state. If such a message is received before the end of the *Cold Start Timeout*, this node will transfer into the *Active* state.

Thus, the node that times out first from the *Listen Timeout* will send the first message, which is an I-frame. Other nodes receiving this message, will set its C-State accordingly and can then change to normal operation

phase where normal messages can be exchanged, i.e., messages with data information.

The basic startup behavior is in reality a bit more complex, for example, TTP/C has a dual communication channel. Such a dual link will affect the startup behavior but in order to compare these startup methods, we will compare with a single channel version. In this comparison we are mainly interested in the average startup times of the protocol, but also the max startup times. However, TTP/C has one extremely unlikely startup case where all nodes enter the cold start mode simultaneously. This will generate a very long startup time, but it occurs with very small probability. We have chosen not to simulate such a case, as it is so very unlikely and it would affect the average time startup very little. If such a case occurred, several TDMA-rounds would be added to the startup time.

In this simulation, we have assumed that the TTP/C nodes are started approximately at the same time and then they run their initialization code. As a consequence, the *Listen Timeouts* of the nodes are started at different times. In this simulation we have therefore assumed that nodes starts their individual *Listen Timeouts* at different times, evenly distributed between time zero and a cycle time, e.g., for a six node system between 0 and 0.2 $ms$.

The startup time and behavior of the TTP/C protocol is mainly decided by the startup timeouts $\tau_i^{startup}$, shown in Figure 2.19, which is unique to each node.

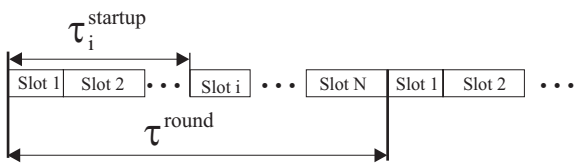


Figure 2.19: The startup timeout.

Together with the TDMA round time $\tau^{round}$, they build the *Cold Start Timeout* and the *Listen Timeout* as follows.

$$\tau_i^{coldstart} = \tau^{round} + \tau_i^{startup} \tag{2.7}$$

$$\tau_i^{listen} = 2 \cdot \tau^{round} + \tau_i^{startup} \tag{2.8}$$

In Table 2.4 we can see the average startup times with their standard deviation as well as the Max startup times for the TTP/C protocol compared to the presented ML- approach.

| Nodes | CT | | TTP/C (ms) | ML (ms) |
|---|---|---|---|---|
| 6 | 512 | Average | 0,52 | 0,37 |
| | | Std. Dev. | 0,05 | 0,03 |
| | | Max | 1,11 | 0,55 |
| 12 | 1024 | Average | 0,97 | 0,68 |
| | | Std. Dev. | 0,08 | 0,04 |
| | | Max | 2,03 | 0,93 |
| 18 | 1536 | Average | 1,42 | 0,99 |
| | | Std. Dev. | 0,10 | 0,05 |
| | | Max | 2,95 | 1,29 |
| 24 | 2048 | Average | 1,85 | 1,29 |
| | | Std. Dev. | 0,12 | 0,05 |
| | | Max | 3,81 | 1,60 |

Table 2.4: Average startup times for TTP/C protocol compared to the presented Message Length (ML) approach.

As can be seen from Table 2.4 our ML approach compares well with existing techniques. It should also be emphasized that the ML-approach does not require any special messages to be synchronized, i.e., it uses normal messages. This means that the communication is established very fast using the ML-approach. When the nodes in the ML-approach reach the Normal state (when we have stopped measuring the time of this approach), half of these nodes have already sent messages with data. This is not the case for the TTP/C protocol, which we stopped the time measuring after the first correctly sent message, i.e., the first I-frame.

The DACAPO protocol [RLST95] is a TDMA protocol where all messages have static and equal lengths, otherwise it is similar to our approach. We will compare the Max startup times of our ML-approach, with the Max startup times of two startup algorithms developed in [Lön99a]. In our comparison with the startup algorithms from the DACAPO-protocol, we have used data from [Lön99a]. Furthermore, we will compare our approach with another existing method, the Lock-step (LS) algorithm [DPC+96], using data from the same source.

We will not go into details of these algorithms, instead we refer to [Lön99a] and [Lön99c]. The two DACAPO startup algorithms are the Zero First (ZF) and Increment/Decrement (I/D) methods and they are developed for safety-critical application. Especially, the I/D algorithm is robust against transient faults. The LS algorithm is highly sensitive to transient faults.

In order to translate those results to our configuration we have adapted the message length to be 1/6 of the cycle time. We have compared our case with 6 nodes and a cycle time of 204.8 $\mu s$. This means that we have adapted the results from [RLST95] to use a message length of 83

| Startup Method | Max (ms) |
|---|---|
| **ML** | 0,55 |
| **ZF** | 2,28 |
| **I/D** | 1,80 |
| **LS** | 0,54 |

Table 2.5: Max startup times for ZF, I/D and LS startup methods (fault-free cases) compared to the presented Message Length (ML) approach.

bits and inter-frame gaps of 2 bits, which results in a slightly smaller cycle time of 204.0 $\mu s$. The results can be studied in Table 2.5. We have only been interested in comparing the max startup times to see whether our new approach is an improvement over our first generation startup methods. As we can see there is a significant improvement, except compared to the LS method. However, this method is highly sensitive to failures and not suitable for safety-critical systems.

In this chapter, we have presented a unique synchronization approach for synchronization in distributed real-time systems targeting safety-critical systems. In the next chapter, we will continue with the time-triggered media access approach and compare its relative performance with the contention based event-triggered media access approach.

# Chapter 3

# Time- and Event-Triggered Multiple Access

THE access to a shared communication media using the event-triggered and time-triggered approaches has garnered its share of strong opinions in the community. Comments such as "event-triggered systems are more flexible and use the bandwidth very efficiently" versus the time-triggered approach which is considered to be more predictable and uses less message overhead. These are generally accepted high-level statements that are taken at face value. However, our interest in comparing the two is to quantitatively assess the actual difference across these paradigms, with the specific objectives being:

- Can we estimate what we lose in flexibility by choosing the time-triggered approach over the event-triggered one?
- How much do we suffer from the loss of determinism by choosing an event-triggered approach?

The effects of choosing one design approach over another are not obvious, especially in a complex system where efficiency, predictability, flexibility, dependability are various dimensions of a design decision. For example, when choosing the event-triggered system we might be prohibited from using the full bandwidth in order to ensure critical message transfers even in an eventful situation, leading to a low utilization of the media.

As we previously have established, time-triggered and event-triggered systems must use some procedures in order to avoid media access contention when sending data. In time-triggered systems, nodes use Time Division Multiple Access (TDMA). For event-triggered communication, the most common technique in use is probably collision avoidance by bit

arbitration, as used in, e.g., CAN [CAN91]. In bit arbitration, a unique identifier is sent in the beginning of each message and determines the priority of each message. Thus, if more than one node sends at the same time the node/message with the highest priority will continue to send while other lower priority nodes defer their transmissions. Bit arbitration limits the possible speed of the bus communication since the bits sent on the bus must be distinct such that a bit collision can easily be detected.

Thus, our specific objectives are:

- We quantitatively establish the conditions under which event-triggered and time-triggered systems are more appropriate to use concerning the amount of transferred data and the response time characteristics.
- We establish schedulability strengths and limitations of each approach.
- Based on the above two facets, we outline suggested domains of strength and weakness for each paradigm along with suggested envelopes of operation.

## 3.1 Related Work

In [TBW95] the authors analyze delays using schedulability analysis resulting from two access methods, timed token passing and real-time priority broadcast bus.

Another similar investigation, [LA99], considers fixed priority scheduling and static-cyclic scheduling on processors and the communication bus. A control application is used as an example where one node reads and prepares sensor data and then sends a message, according to one of the scheduling techniques. This message is sent to other nodes that use the received information to calculate the actuator data. They also study the effect of using a global time. With this setup, the minimum and maximum delays and jitters have been calculated for the two scheduling techniques. In this study, we focus more on the average delay of messages due to the communications system. Also, how the average delay differs under different working conditions.

## 3.2 System and Task Model

In this section we will describe the system and task model used in this chapter. We use the basic system model described in Chapter 1.1, concerning both the time-triggered and event-triggered approach. In this

section we will give some further details that are specific and important for the simulations in this chapter.

It is assumed that communication takes place between two tasks $\tau_i$ and $\tau_j$. The message $m_{i,j}$ has length $e_{i,j}$. The message size is measured in bytes (1 byte=8 bits). The tasks are periodic and must execute within the time interval given by the period. The communicating tasks are assumed to be located on different nodes so the communication will take place over the communication media. In the schedule, it is assumed that a message must be sent after the completion of $\tau_i$ and be received before the start of $\tau_j$. These messages are called *periodic* messages and their arrivals are known beforehand. There can also be *sporadic* messages (which do not have sender and receiver tasks) whose arrivals are unknown. Periodic messages must always be handled (on time) whereas sporadic messages are handled when possible. Periodic messages do not have explicit deadlines but must be delivered in time for the receiver task to be able to meet its deadline. Message transmission is always non-preemptive. In addition, the message transmission must comply with the media access method.

### 3.2.1 Time-Triggered Approach

In order for the system to synchronize or a node to resynchronize, messages must be identifiable to find the current position in the communication schedule. Thus, the overhead associated with time-triggered communication concerns the identification of the sending node. We therefore assume, a node ID-field of one byte , i.e., the size of a message will be $e_{i,j}+1$. Sporadic messages must be sent in a slot that belongs to the node where the (fictive) sender process executes. For a periodic message, the sending task must finish before the node's send-slot and the receiving task can start after receiving the data sent in that slot. This is a natural way of sending periodic messages but is limited in flexibility when it comes to sporadic messages.

In this work, we use a simple and natural way of sending sporadic messages in the time-triggered architecture (TTA). One frame, i.e., message data including overhead, is sent in each time slot. Frames are used as containers for both periodic and sporadic messages, they have one static part assigned for periodic messages transfer, i.e., the same message data is transferred periodically in that part. Then we use the second part of the frame for sporadic message transfer. As we do not know the arrival times for sporadic messages, we use a node local queue where all new sporadic data is queued when arriving. Messages are queued based on priorities. When its time to send a frame, as many messages as possible

are packed in the sporadic part of the frame.

### 3.2.2 Event-Triggered Approach

In our Event-Triggered Architecture (ETA), we assume that nodes can resolve potential collisions by the inclusion of priorities in the message-frames. Thus, if two nodes start to send at the same time the node with the highest priority message-frame will send and the other node will withdraw.

We have assumed that periodic messages have the highest priority and sporadic messages have lower priority. Compared to a time-triggered protocol, the communication overhead now also includes identification of a message. Therefore, in our investigation the size of a message will be $e_{i,j} + 2$, where one byte is for message ID and one byte for node ID.

## 3.3 Communication Issues

The system we assume consists of a system input value $s$, e.g., a sensor value, received by a node $n_a$. This input value is prepared by a control task at node $n_a$ for transport to a receiving node $n_b$. This node will contain a task that will make some final calculations on the message/value and will finally provide the output device, e.g., an actuator, with its value.

Most control applications are periodically sampled, thus this task chain is activated periodically. We assume that the tasks are activated by the OS, starting with the first task in the chain at node $n_a$. The task activations have a small jitter, where jitter is the difference in time between start or finish time of different invocations of a task, see Figure 3.1. A number of reasons can cause this, for example, clock drift or different delays in interrupt routines. This jitter is normally very small compared to difference in the execution times of the tasks, which then contributes more to task jitter. If this task sends a message on the media, it will be queued with the same kind of jitter. The jitter can be smoothened by sending the message at a time when the task has finished with a high probability. However, this smoothening comes at a cost of an increased delay, as on average the wait is longer to send the message. To summarize, the following factors are considered as the main contributors to delay and jitter.

- Task activations
- Task execution time
- Message output queue

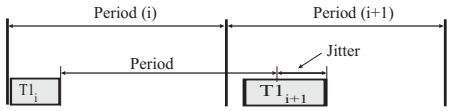


Figure 3.1: Jitter of task finish.

- Message transfer
- Message input queue

In real-time control systems, it is important to have small delays and jitter. In this chapter, we have focused on the message output queue, as this is where the communication access method has its main impact.

### 3.3.1 Periodic Messages

Periodic messages are often used in computer control application. Most such applications are sensitive to delays and variations in delay, i.e., jitter. In this section we will shortly describe the upcome of delay and jitter in our two communication architectures.

**Delay in TTA**: Delay is fixed, as long as messages are generated periodically and in synchrony with the communication system.

**Delay in ETA**: To get fixed delays here we need to ensure that the sending node gets access to the media at periodic times. Delay occurs when a message has to wait for higher priority messages to be transferred. Delays can also occur while waiting for a lower priority sporadic message to finish its transfer.

**Jitter in TTA**: Normally TTA prevents jitter from occurring, as messages are transferred at fixed times.

**Jitter in ETA**: Jitter is the difference in delay between instances of a periodic message.

### 3.3.2 Sporadic Messages

Jitter is not applicable to sporadic messages, by definition, as they do not have an explicit expected arrival time.

**Sporadic Messages in TTA**: In our approach, we pack sporadic messages in the same frames as the periodic messages, thus they are sent in the same slots. The delays for sporadic messages in a time-triggered architecture are dependent on a number of factors: the time until the next send slot, how sporadic data is transferred using the broadcast media, and amount of bandwidth allocated to sporadic messages.

The *minimum* time it takes to access the media, occurs when a message is generated just before the sending node should send its sporadic traffic. The *maximum* delay occurs when a message is generated just after the send point of the sporadic data. As the message was generated after this point, it has to wait an additional TDMA-round.

**Sporadic Messages in ETA**: A sporadic message can immediately access a media, if the media is free and no higher priority message tries to send simultaneously. However, there is no theoretic upper bound on the access time, as higher priority messages can continuously prevent a message from media-access.

Due to the strong interdependencies of the messages and their distributions, we have not at this time derived an analytical expression of the average delay for message transfer. At this point, we want to get an understanding of the behavior of these systems which is easier to obtain with simulations. The simulations also provide important insights into the factors that affect the performance.

## 3.4 Simulation Setup

To do this comparison, we have made two investigations. One where we check the behavior of time-triggered and event-triggered communication under different communication loads. In the second, we have checked how an event-triggered and time-triggered system handles different task sets, i.e., how easy it is to schedule different tasks and messages using these different approaches.

In order to get synthetic task-sets and sets of sporadic messages, we generated those randomly using a uniform distribution. These tasks and sporadic messages were generated in two steps: (1) a period was randomly generated and (2) within each period an activation time of the task/message was randomly generated.

Thus, the generated periods control the basic rate of sporadic messages and tasks. These sporadic messages together with the length and period of each message, provide us with the bandwidth these messages are likely to occupy, provided they get access to the media. Furthermore, each message and task were assigned a priority that is used to prioritize between messages accessing the media and released tasks. Thus, each message and task are assigned three properties (1) a period (2) a length and (3) a priority, i.e., a triple $< period,\ length,\ priority >$ that represent the characteristics.

The system is run with different communication loads in order to compare the time-triggered and event-triggered approach. The generated communication load, consists of both periodic and sporadic messages. Pe-

riodic messages are generated at a constant rate, while sporadic messages are generated with a varied rate. Thus, the amount of periodic messages is fixed and the communication load is determined by the amount of sporadic messages. That is, the *generated sporadic traffic* is randomly generated to achieve a certain load. The *load* is controlled by the message length, number of messages and the basic period of messages. Specifically, we have defined *load* as:

$$Load = \frac{Generated\ sporadic\ traffic}{Available\ sporadic\ bandwidth}$$

The available bandwidth is the amount of data that can be transferred per time unit on our media. Thus, the *Available sporadic bandwidth* is what is left after the sporadic traffic has reserved its bandwidth share.

### 3.4.1 Task and Communication Scheduling

We start by investigating the schedulability of tasks and messages for the different communication paradigms. The scheduling considers the tasks in the system as well as the communication amongst them. This is a difficult problem in itself and there is a multitude of research in this area, e.g., see [PSA97, TBW92]. The purpose of this investigation is to find out how the scheduling is affected by the communication and which communication parameters are sensitive for the schedulability of the system.

In a TTA, periodic messages (and tasks) follow a predefined schedule and sporadic messages are included when there is slack available. Hence, sporadic messages will not affect the pre-scheduled tasks and messages or cause them to miss deadlines.

In a ETA, the scheduling of tasks and messages is performed on-line. Tasks are normally preemptive, but messages sent on the media are not preemptive and can delay a higher priority message, which may lead to a missed message deadline or task deadline.

#### Scheduling

We will here investigate how the communication scheduling affects the ability to handle a mix of periodic and sporadic messages. In such an investigation, it is important to know how the tasks in the system are scheduled, since this also affects the communication. In general, the choice of media-access method reflects the intended use of the system. That is, a time-triggered approach is focused on predictability whereas the event-triggered approach offers flexibility. It would then seem natural that the motivation for choosing a particular communication approach is

also applicable for how the tasks should execute, i.e., how to perform task scheduling. Hence, when using time-triggered communication, the task schedule constitutes a static time-table dictating the start times for the tasks which are not affected by any sporadic messages. Similarly, the task schedule for the event-triggered architecture is generated dynamically by considering tasks priorities. (The priority of a task equals the priority of its message.) This also means that tasks and periodic messages may fail to meet their deadlines due to interference of sporadic messages.

### Experimental Setup

The purpose is to investigate how the sporadic messages affect the periodic messages and in turn the tasks, when using the TTA and the ETA respectively. This is done by generating synthetic task sets consisting of both periodic and sporadic messages. For TTA, a task set is then examined by *(i)* finding an off-line schedule for the tasks and periodic messages *(ii)* simulating the arrival of the sporadic messages and measure the number of sporadics that fail to be sent on time. For ETA, tasks and messages (both periodic and sporadic) are handled in the same step by simulating an on-line fixed-priority scheduling algorithm. Hence, the quality measure also includes the number of deadlines missed by the tasks and periodic messages. For this part of the simulations we have used a constraint programming framework that were previously developed [EJ01].

We generated three studies representing systems with 6, 12 and 18 nodes. In each study we generated task sets with varying sporadic communication load, see Table 3.1. The variation was obtained by successively increasing the message sizes while the number of messages remained constant.

The deadline for a task equals its period. Communication between tasks take place in pairs and all sender/receiver tasks on a node communicate with tasks on the same node. To get the load evenly distributed in the schedule, the tasks also have randomly generated activation times. For each experiment, 20 task sets were generated.

### Experimental Results

In the first experiment, we have investigated how well the ETA and the TTA can handle sporadic messages. If a sporadic message cannot be sent in such a way that it is received before its deadline, it is regarded as missed. The same concept applies for periodic messages as well as task executions. Figure 3.2 shows the number of missed messages for the TTA and the ETA. As can be seen in the figure, the ETA is able to accommodate a larger proportion of sporadic messages than the TTA.

| **Number of nodes** | 6 | 12 | 18 |
|---|---|---|---|
| Cycle length | 960 | 1920 | 2880 |
| Number of tasks per node | 8 | 8 | 8 |
| Number of sporadic messages per node | 4 | 4 | 4 |
| Period of sporadic messages | 240 | 480 | 720 |
| Task execution times | 20–60 | 60–100 | 100–140 |
| Periodic message sizes (bytes) | 8–16 | 8–16 | 8–16 |
| Sporadic message sizes (bytes) | 1–20 | 1–20 | 1–20 |
| Slot size (bytes) | 32 | 32 | 32 |

Table 3.1: Configuration parameters for the task sets. The time unit, is the time it takes to transfer one byte.

However, as the load increases, the gap between ETA and TTA decreases. The explanation for this transition is that at low load, the ETA has the flexibility to use the slack for any node. This is not the case for the TTA, thus the number of missed messages grows almost linear with the load. In contrast, when the load increases, ETA still tries to accommodate all messages before their deadline. However low priority message can now easily be delayed by messages from all nodes, such that they miss their deadline. This is an effect which increases with load. This problem occurs because in the ETA the communication queue is *global*. In the TTA this effect is limited since the communication queue is *local* and thus fewer messages are affected. Hence, the TTA avoids the "domino-effect" of missed deadlines found in the ETA. As seen in Figure 3.2, the transition occurs at a load of approx. 0.6–0.7. However, this can be explained by the fact that the task executions constrain when the (periodic) messages may be transmitted which makes it harder to fully utilize the media.

For the TTA, note that in our experiments the sizes of the messages are rather large compared to the size of the slots. This means that sporadic bandwidth may be wasted due to that many messages will not fit into the remainder of the designated slot. Hence, if we had varied the load by increasing the number of messages instead of increasing the message size, the performance of the TTA would probably improve.

We have assumed small frame overhead for addressing in both ETA and TTA, i.e., approximately 1 and 2 bytes for TTA and ETA, respectively. If we should include other overhead, e.g., for checksums etc., we would likely effect the ETA negatively. With $n$ nodes, and assuming a frame overhead of $h_{TT}$ for TTA and $h_{ET}$ for ETA. The total overhead,

$H$, for TTA during a TDMA cycle is:

$$H_{TT} = n \cdot h_{TT}$$

We assume similar condition for the ETA and assume one periodic frame per node. In addition, we send $k$ sporadic frames, during an equivalent time of an TDMA round in the TTA. The total overhead in the ETA is then:

$$H_{ET} = (n + k) \cdot h_{ET}$$

Thus the more sporadic frames that are sent, the more the efficiency of the ETA will decrease compared to TTA. This effect can naturally be decreased by combining more messages in each frame, as done in the TTA, although that could reduce the flexibility.

In the ETA we get the additional information about the number of missed periodic messages and task deadlines. Recall that for the TTA there will be no such misses but for the ETA a substantial amount of deadlines and periodics are missed. For instance, in Figure 3.2 the amount of missed messages in the ETA curves that corresponds to periodic messages, ranges between 5% and 15%. Recall that a missed periodic message also means a missed (receiver) task deadline.

### 3.4.2 Communication Load

In this section we present an investigation of the average delay for a message, from its release time until it is actually transferred. It is important for all communication systems to achieve good response times and to be able to work in a real-time environment. Compared to the previous investigation, we have removed the impact of tasks, and we now only look at the effects of communication. The result is equivalent to assuming tasks with very short execution times, thus the system is only limited by the communication. In these simulations we have used Matlab as the simulation tool.

In the following sections, we present simulations of the event triggered and time-triggered system. The simulations has been conducted in three basic steps: (1) In each simulation we have randomly generated a message set, that is, for each message in this set the triple $< period,\ length,\ priority >$ is generated as described in Section 3.4. These messages have been generated such that we get the desired message load. Furthermore, messages are distributed among the nodes such that they all have approximately the same average load on the media. In step (2), the release instances of the messages are generated. This will determine at which points in time all sporadic messages are sent. The

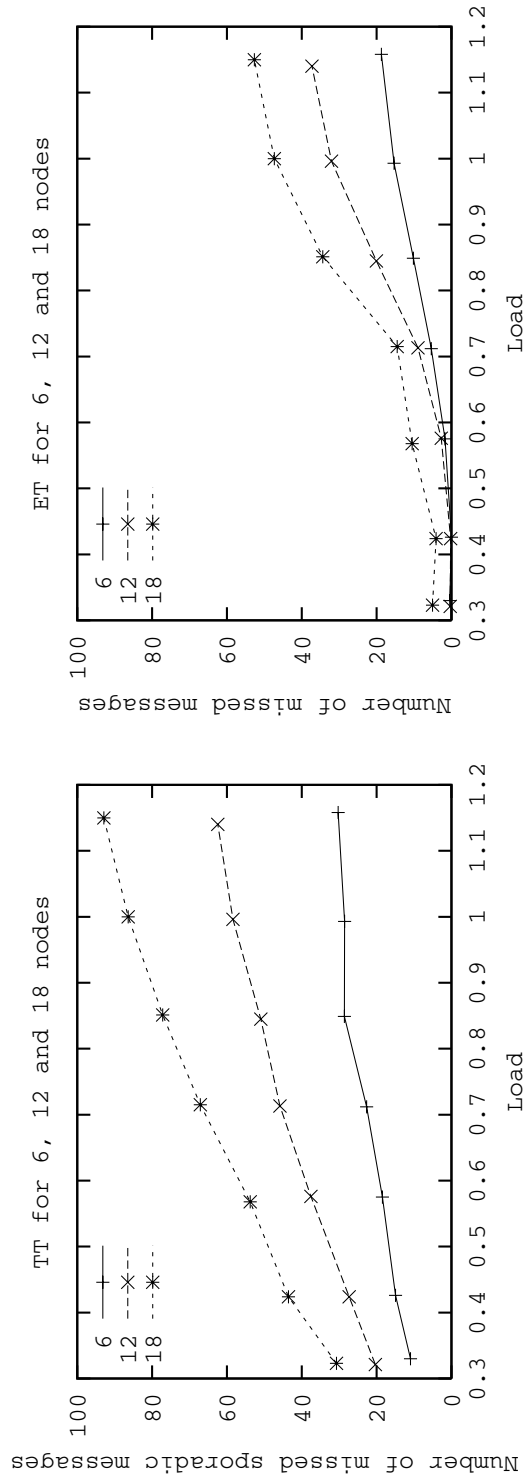


Figure 3.2: Event-Triggered and Time-Triggered (30% periodics, 30% task utilization).

release times for the sporadic messages are randomly generated using the triple $< period, length, priority >$. The send times for periodic messages are already fixed by their periods. In the final step (3), the simulation is run and simulation data is collected, the nodes will send their messages at the times calculated in previous steps and will be sent using the active access method, i.e., the event-triggered or the time-triggered approach.

### Simulation Time

As there is no natural termination point in this type of simulation, we want to determine a point in time where the behavior of the system has stabilized. We have therefore started by investigating the effect of the simulation-time, i.e., the period of time the simulation is run. A system with six nodes were used, and we run this system for different time-periods and compared the result. This is shown in Figure 3.3 where we have run the same six nodes system for event-triggered and time-triggered communication but varied the number of runs per simulation. As we can see in Figure 3.3, there is little difference between the event-triggered curves similarly there is little difference between curves from the time-triggered system. Thus, the system is well stabilized after 100 rounds, in the sense that the average delay of messages is not affected even if the system is run for a longer period of time.

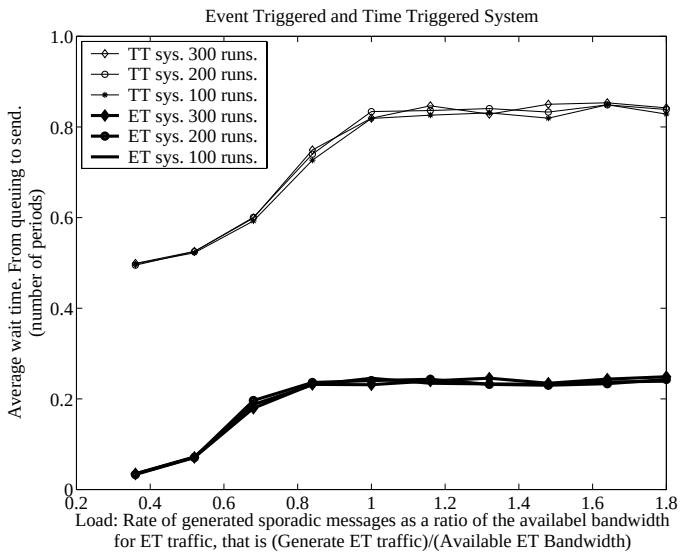


Figure 3.3: Changes in average message delays when increasing time from 100 to 300 rounds.

### A First Comparison

In this section we present the behavior of the time-triggered and event-triggered approach in our simulations. In Figure 3.4, we can see the results after simulations with six nodes and where the bandwidth is divided equally between periodic and sporadic messages. There are two sets of curves, corresponding to right and left axes respectively. The curves belonging to the left side axis show the average delay of messages. That is, from the time they are released until they are sent on the media. This time corresponds to the time in the output buffer. The right axis show the amount of messages not sent. That is, a message $a_i$, i.e., instance $i$ of message $a$, is deleted if the next instance, $a_{i+1}$ arrives before $a_i$ gets access to the media. Each deleted message instance is counted. To be independent of the simulation run time we divide this number by the number of simulation rounds, for the periodic messages. The result is presented as "not sent messages". This is a good measure of how a particular approach manages to transfer messages at a given load. If no messages are deleted, all generated messages are transferred.

It is important to note that in the time-triggered case, periodic messages will not contribute to the average delay, as periodic messages are sent in predetermined time slots resulting in no delays. Furthermore, they will never be deleted in the TTA approach as they have their pre-assigned slots. This can happen in the ETA approach, but with low probability. When comparing these results, note that periodic messages will be delayed in the ETA. Low priority sporadic messages, cannot be interrupted even by higher priority messages and high priority messages can be delayed by low priority messages.

In Figure 3.4, the average wait time for the time-triggered case starts at half of the period time of the periodic messages, i.e., corresponding to the TDMA round. However, when the loads have increased to around 0.6–0.7, there is a point where the delay starts to accelerate. This increase, is temporary and the delay levels already after a load of 1.

The reason for this behavior is that most sporadic messages are sent with a relatively short delay or completely miss their opportunity to be sent, i.e., the messages are deleted. Deleted messages will not increase the average delays, thus for the TTA the average delays continue to be rather low even when the load is increasing. Instead of a large increase in delay, we pay in an increasing amount of deleted messages as the load goes up.

ETA has a very low average wait time especially at lower loads (approx. $< 0.6$), where messages seldom have to wait to access the media. Although a slight increase, it continues with a low average delay. This

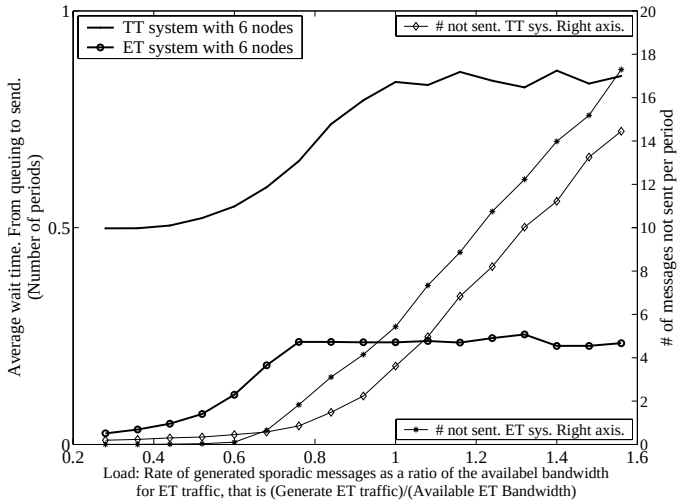


Figure 3.4: The average send delay of the messages. Note that, in the time-triggered case, periodic messages never contribute to this delay.

emphasize the flexibility of the ETA approach where any node can use the media when it is free.

We can however see, that the number of deleted messages starts to grow earlier than in the time-triggered case (approx. $< 0.6$). We can basically see lost messages as messages which missed their deadline, and we saw a similar phenomenon in the simulation with tasks. The main reason for this is that the lowest priority starves from bus access in combination with the ETA has a higher message overhead.

In Figure 3.5 we show the standard deviation of message delays, and both event-triggered and time-triggered have a similar form. There are small variations for limited loads but these increases with increasing load for both the systems. We can note that this increase starts earlier then in Figure 3.4. Also in this case we can see that the ETA increases first followed by the TTA.

**Summary with comparative pros and cons**: The *Event-Triggered* approach has very short average delays with small variations at low load. Almost all the generated messages get sent under a load of approx. 0.6. However, we should note that high-priority messages can be delayed, even by lower priority messages. The delays increase at higher load (approx. $> 0.6$) but stays short for all messages that are sent. The variation of the delay grows earlier, around a load of 0.5. Lowest priority messages, never get access to the media at a higher load.

The *Time-Triggered* approach has no delay of higher priority messages, i.e., periodic messages. However, the average delay is approxi-

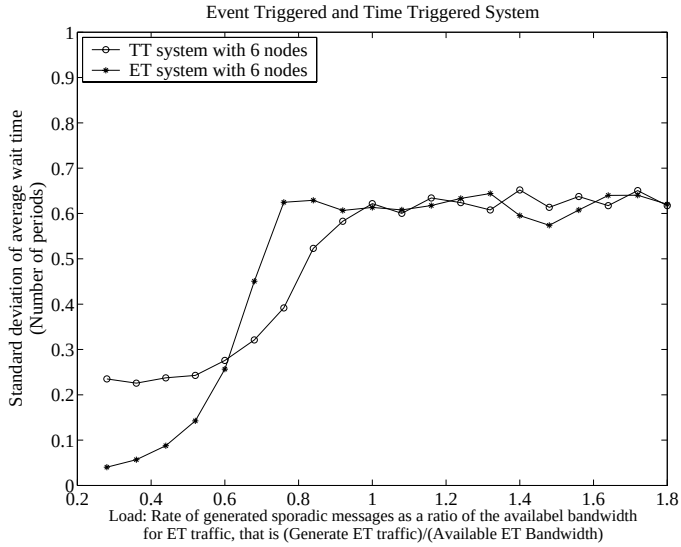


Figure 3.5: The standard deviation of the send delay for the messages.

mately half of a TDMA round, already from the beginning. There is only a small increase in the average delay at high load. In the TTA there is a equalizing effect, as all nodes have a preassigned part for sporadic data in frames. Thus, even lower priority sporadic data gets access to the media.

### Number of Nodes

In this section we have varied the number of nodes in the system and studied changes on the average message delay. We have changed the number of nodes between 6, 12, and 18. Thus in Figure 3.6 we show the trends when we increase the system size, both for time-triggered system and in an event-triggered system.

In Figure 3.6 we can see that the trend scales with the system size. The only parameter we have changed is the number of nodes. However, we use a fixed average size on messages and this will affect the period for sporadic messages. In the time-triggered case this implies also the size of the TDMA-period.

The time-triggered system will in this case get larger delay as the system size increase. This is natural as the average delay is mainly dependent on the TDMA period.

The event-triggered system has still low average delay as long as the load is low. The delays will however grow earlier and at high load the average delays are still low for messages accessing the media.

The amount of missed messages is not affected by the system size.

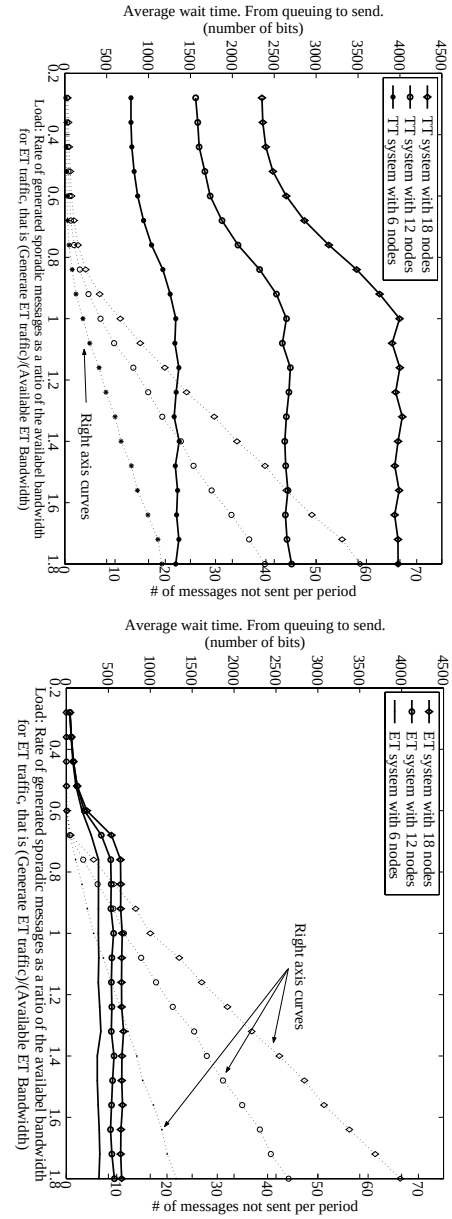


Figure 3.6: The number of nodes in the system is changed from 6, 12 and 18 nodes.

### Preassigned Share of Periodic Messages

In this section we investigate how the system behaves and how the response times change when the share of periodic and sporadic messages are changed. We have varied the amount of bandwidth that is preassigned for periodic messages, the remaining bandwidth is free for use by sporadic messages. The periodic part has been 30, 50 and 70 percent of the total TDMA round. The random generation of sporadic messages has been adapted to the bandwidth situation. For example, when 70 percent of the TDMA round is assigned to periodic messages, it limits the amount of large messages that can be sent. In Figure 3.7 we show the result from varying the amount of preassigned periodic traffic.

## 3.5 Discussions

In this section follows a discussion about the results. We should first note that these results are based on simulations with randomly generated load. These results are naturally not adaptable to any real-life situation, in fact it is easy to create system scenarios favorable for either of these approaches. However, the basic properties of the generated load can be found in many real-time systems. For example, systems with high priority control loops generating high priority periodic messages and where other system parts generates sporadic messages, e.g., with diagnostics and status information.

We argue below that the results point toward certain design decisions, which means that these approaches have certain inherent qualities which make them good in a certain context. However, we do not argue that it is impossible to design a system without following these results.

We also want to note that we intentionally have tried to make extreme cases of the ETA and the TTA. For example, we could easily design a event-triggered media access method, based on frames with fixed size and assign priorities to these frames such that it would be very similar to a TTA. Naturally, the result of simulating such a system would provide little additional information as it would basically behave as the TTA.

• In this work we have confirmed that ETAs provides a flexible way of transferring messages and that it has short average delay for sending messages. The delays are low for all messages that are actually sent. However, the number of missed/deleted messages increases when the load is greater than (approx. 0.7), as the lowest priority messages never get access to the media. We have a similar situation when considering tasks and messages with their deadlines. Then, there is a noticeable amount of missed deadlines already at a load of 0.6–0.7. This is also confirmed

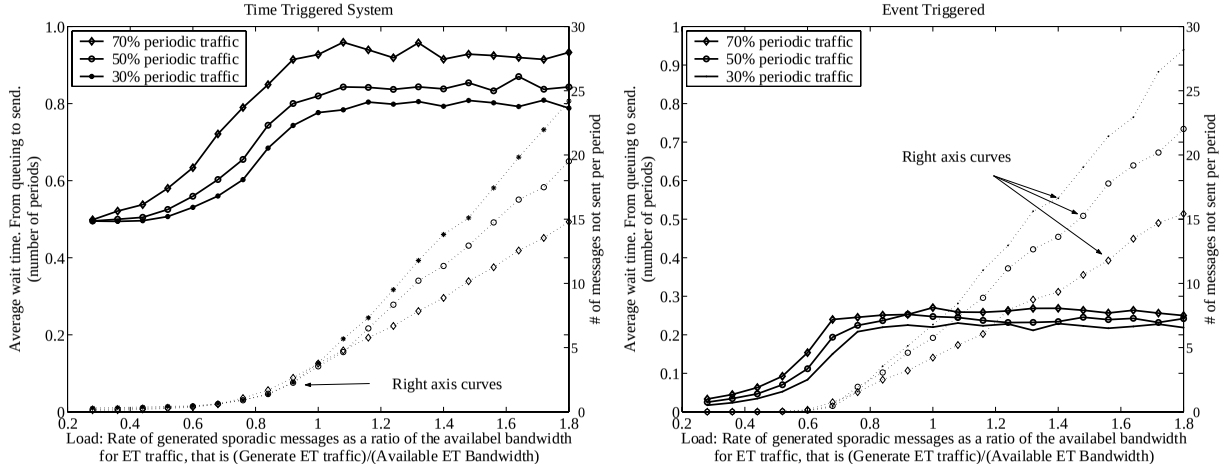


Figure 3.7: Varying the amount of preassigned periodic bandwidth in an event-triggered and time-triggered system.

by the pure communication investigation as the standard deviation starts to increase around the same load. Thus, the fact that there is a global queue together with big variations in delay, larger message overhead and more messages cause a lot of missed deadlines in the ETA.

• For the TTA, the average delay is approximately half a TDMA-round, which is large compared to the ETA. Note, that periodic messages in TTA systems have zero delay and only sporadic messages contribute to the average delay. There is also a higher amount of lost messages compared to the ETA when tasks are involved, i.e., when short deadlines are used for the sporadic messages. In the pure communication case, short deadlines are not used and we can see that fewer messages are missed for the TTA, at higher load, than the ETA. Thus, short deadlines on sporadic messages effects a TTA negatively. With not so short deadlines of sporadic messages, it is noticeable that this very basic approach of transferring sporadic messages still handles sporadic traffic surprisingly efficient, in the sense that few messages are missed/deleted.

Thus, the predictability of a TTA comes at a cost of longer delays, but with less variation in these delays. However, when sporadic messages deadlines are relatively long, there is a smaller loss in bandwidth efficiency compared to the ETA.

This basic approach of sending sporadic messages works with very small or no changes in any TTA. In our future work we will look at more efficient ways of transferring sporadic and event-driven data using TTA.

• As long as a system runs with a load below 0.6–0.7, there is basically no reason for choosing TTA. The ETA imply short message delays and that basically all messages are transferred. Even variations between delays are small and the number of missed deadlines few below this threshold. The only reason for choosing a TTA is if we have high requirements on predictability, e.g., dependability.

The capacity of a ETA to handle sporadic traffic is one of the main contributing facts to why ETA is considered flexible. However, the TTA can also handle sporadic messages well, as it is not so sensitive to traffic bursts and benefits from the fact that it has less overhead than the ETA. As we have seen in the figures, both TTA and ETA have similar amount of lost messages. Thus, when considering flexibility of the ETA we would rather point to the ease of integrating new nodes etc.

• In the case of varying load/bursts traffic or high load, i.e., the load is often above 0.6–0.7, a TTA should be considered, especially for real-time systems. However, based on our result we can also state that the only reason for choosing the ETA below the 0.6 threshold is if there is need for small average delays of sporadic messages.

## 3.6 Conclusions

In this chapter we have studied two contemporary media-access methods, the time-triggered approach and the event-triggered approach. In the first simulation, we studied a system where we considered both tasks and communication. In the second, we studied the system by looking exclusively at the communication media. To conclude the main results:

- Event-Triggered Architecture
  This is a very beneficial approach if short delays are important and the load is kept below approximately 0.6–0.7. The main properties are:

  \+ Short average delays for sent messages.

  – High-priority (periodic) messages are delayed. Starvation of low priority messages at higher load. The variation in delays grows early around a load of 0.5.

- Time-Triggered Architecture
  This is a good approach for all tested loads, as long as the requirements on delays for sporadic messages are not very high. In such a case, the TTA is preferable only for loads above approximately 0.6–0.7. Main properties:

  \+ No delays of high priority (periodic) messages. Few missed sporadic messages.

  – Sensitive to short deadlines of sporadic messages. High average delay for sporadic messages, (half a TDMA-round).

In this chapter we have tried to give a system designer more knowledge how the choice of event-triggered or time-triggered as basic media access affects the communication. More specifically, how it affects the average delays and transfer capabilities. In the next chapter, assume that we are really not prepared to trade the predictability of the time-triggered approach. At the same time we desire the fast handling of the sporadic messages, that the event-triggered approach provides. Thus, in the next our goal is to combine the chapter

# Chapter 4

# Event-Triggered Channel on Time-Triggered Base

IN the previous chapter, we studied the differences between event-triggered and time-triggered communication with regard to average delay and schedulability. It is clear that the event-triggered approach has short delays for sporadic messages compared to the time-triggered approach. However, the time-triggered approach still manages to transport a comparable amount of traffic under the condition that few deadlines of such sporadic messages are shorter than half a TDMA-round. We also assumed a very basic mechanism for transferring sporadic messages via a time-triggered channel.

In this chapter, we will assume that for reliability and safety reasons we require the predictability provided by time-triggered communication. However, we want to improve the handling of sporadic messages that is currently limited in the time-triggered approach. We argue that the best way of maintaining the predictability attribute is to start with a time-triggered base and build the necessary event-triggered channels on top. Therefore, we investigate and introduce methods for combining the benefits of both time-triggered and event-triggered communication.

We emphasize that our primary driver is to ensure that none of the essential properties of the time-triggered paradigm, such as predictability, are in any way compromised over the process of adding handling of sporadic traffic. Time-triggered communication has established properties for provision of safety-critical services and we consider these as inviolable over our process of extending and modifying TT services to handle ET traffic. In this chapter, we suggest a variety of methods based on the time-triggered communication base, on which we add functionality for improved handling of sporadic messages.

This chapter consists of two parts. The first suggests approaches

for improved handling of sporadic messages, both high-priority and low-priority messages. We assume a basic Time Division Multiple Access (TDMA) system as described in the introduction, Section 1.1.2, but we adapt the system model where necessary, to achieve our goals. In the second part, we have limited ourselves to a specific communication protocol, the TTP/C protocol [KG94, TTP99]. Our goal is to derive an adaptation to the protocol that improves the efficiency of sporadic message transfer. It is important to achieve this without any disruption of the time-triggered traffic or the underlying periodic behavior. As this protocol is directed towards safety-critical systems, much effort and work has been expended in ensuring its correct behavior. Thus, our solution must work with very limited changes to the protocol specification so as to minimize any subsequent re-verifications. We especially try to modify only those aspects within TTP/C that do not change any behavioral characteristics of TTP/C. For example, if an existing TTP/C frame is utilized for handling sporadic messages in place of periodic traffic, from the TTP/C protocol viewpoint, there is no change to the frame specification of the protocol.

## 4.1 Lower Priority Non-Periodic Messages

In this section, we study scenarios where we have low-priority sporadic traffic that must be scheduled "over" the time-triggered traffic. The intention is to make as many sporadic messages meet their (soft) deadlines as possible.

### 4.1.1 Pre-Scheduled Slack

This is the method used in Chapter 3, where each node schedules some slack at pre-runtime, which is pre-destined for sporadic/event triggered traffic from that node, see Figure 4.1. Thus, a node fills a pre-defined part of the message-frame with periodic data, and when a node has sporadic messages to send, it will pack as many as possible in the remaining part of the frame. The size of the periodic part is determined pre-runtime, as the frame sizes are fixed. Thus, the size available for sporadic messages is also fixed. The properties that follow this method include:

- The sporadic data normally needs addressing information, such that the nodes can distinguish between different sporadic messages from a node, i.e., if there they receive several from the same node. Thus, each data entity must include an address or ID. Please note that for periodic data, address information is implicit as this data is statically scheduled.

- The length of a message-frame is constant and the same as the slot-length. A node cannot adaptively change the amount of data it sends. Furthermore, a node cannot increase or decrease the amount of "event data" to send, as the slack is statically scheduled for each node.

- A node cannot utilize unused slack of other nodes. This implies that there is no global priority on sporadic data. The priorities among the sporadic messages are handled internally in each node, i.e., each node handles its own slack.

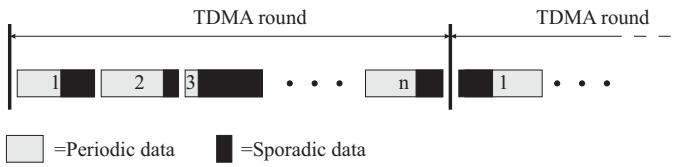


Figure 4.1: The TDMA approach, with slack, for event-triggered/sporadic traffic.

The drawback of this approach is that message-frames are fixed, both in position and length. This prevents a node from adapting to differences in the need for transferring sporadic data, a node cannot share the unutilized portion of the media-access time that is pre-assigned to it.

### Improvement suggestions

Is there any way of reducing the described limitation, i.e., the inability to dynamically adapt to different bandwidth requirement? As the frame sizes are fixed, only node-internal measures can be taken. However, one possibility is to permit a flexible assignment of the frame size for periodic and sporadic messages. This would, however, necessitate that a periodic message at some occasions would be delayed/skipped to make way for a sporadic message. Naturally, this only applies to situations where the system is expected to perform such prioritizing among the messages, and must be decided by the system designer.

For example, consider the case where a node $b$ encounters a situation where a sporadic data $s$ has higher priority than one of its scheduled periodic messages, $p$. If $s$ does not fit within the normally scheduled "slack", node $b$ can then decrease the amount of periodic data by delaying or skipping $p$, such that the important sporadic data, $s$, will fit. The cost of this improvement is increased complexity for handling varying shares

of periodic and sporadic data. Furthermore, there is an overhead of one or more bits for indicating the current load of a message-frame.

### 4.1.2 A Mixed Access Method

In this method periodic data is scheduled first and no slack is used in individual nodes. The TDMA-round is divided in two parts, one where nodes access the bus by TDMA and a second part for sporadic data, using an alternate media access method. Thus, in each TDMA-round there will be a certain free time not occupied by statically scheduled data, which is free for sporadic traffic. The drawback with this method is that it needs a special bus access procedure for the sporadic messages. Examples of possible bus access methods include:

- Minislotting. The combination of TDMA and minislotting is, for example, used in Arinc 629 [ARI95].
- The use of bit arbitration.
- The use of tokens.
- A Media Master who sends the schedule in the periodic message. For example, each node asks/requests a message slot in the "sporadic area" in the periodic message, the last node who sends in the periodic area is a "master" and decides when and how much bandwidth each node is assigned. This would require that all nodes send a message in each TDMA-round such that they can demand a piece of the "sporadic message area".

This method can have variants where a TDMA-round is divided into a number of periods, with periodic and sporadic areas that alternate. Each periodic area would end with a node functioning as a "master" in the following sporadic area, which will be partitioned among the nodes, by the master node.

## 4.2 High-Priority Sporadic Messages

In this section, we will assume that we have sporadic messages with high-priority, i.e., higher than the periodic messages sent with the time-triggered channel. We describe how the normal periodic schedule can be interrupted by higher priority messages. Thus, if we have a normal TDMA system running and we suddenly want to interrupt this schedule with a very high-priority message, how can this be accomplished?

Ultimately, we would desire the following from such an interrupt function:

- Interrupt the normal TDMA-round at any time.
- Continue with the TDMA-round after the high-priority message has been sent.

We basically have two ways of transferring an interrupt request, either *explicitly* by sending on the channel, or *implicitly* by not sending, i.e., using delays like the minislotting approach. Furthermore, we can choose to always interrupt after a message or, in case of haste, interrupt the sending message, in which case we must garble the sending message such that the sender withdraws.

After a TDMA-round has been interrupted by a high-priority message, the normal TDMA communication must resume. There are three main candidate approaches for implementing this:

1. Maintain the timing of the normal TDMA schedule, see Alt *a* in Figure 4.2. This means that after the high-priority message the next message in time to send according to the global schedule will be sent. This method does not introduce any jitter, i.e., varying delays of the periodic messages in the TDMA-schedule. However, one or two messages may be lost while the high-priority message is sent.
2. The interrupted node can continue, i.e., re-send its interrupted message, see Alt. *b* in Figure 4.2. With this approach we do not miss any messages, but all messages will be delayed.
3. In this approach the TDMA schedule is restarted, see Alt. *c* in Figure 4.2.

There must be an understanding among nodes when they should start after a high-priority message. Not only which node should continue, but also at what point in time. This can be important for the synchronization of nodes and local clocks. We also assume that the length of high-priority messages-frames are fixed. Hence, there is a fixed time after all high-priority messages transfers before the normal TDMA-schedule can progress, (according to one of the approaches in Figure 4.2). In such a case, all nodes will know when to expect the normal communication to continue.

Another approach would be that the node next in line to send would listen to the media and start sending as soon as the high-priority message finishes. However, this would introduce an uncertainty of when the communication continues, i.e., a varying delay of regular messages when higher priority messages are sent.

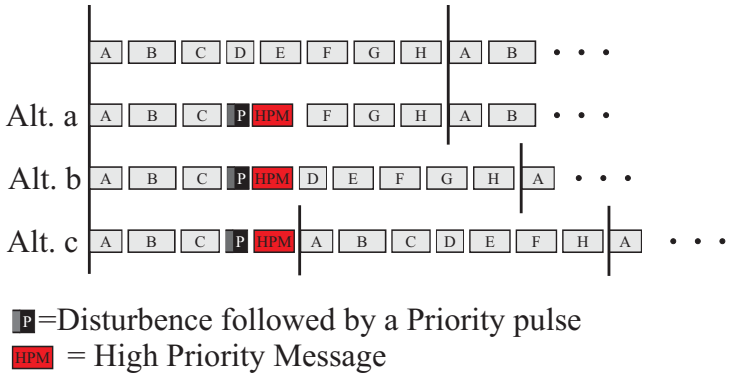


Figure 4.2: A node with high-priority messages interrupts the normal TDMA-round by a reset pulse. Alt. *a*, keep timing undisturbed; Alt *b*, interrupted message is re-sent; and Alt. *c*, restart the TDMA-round.

### 4.2.1 Reset Pulse

In our first suggested method, a reset pulse is used to interrupt the TDMA-round. This reset pulse consists of two parts; one part with the main purpose of disturbing the current sender, such that it stops sending. The second part consists of a very short message indicating that the interrupt was an intentional interrupt, due to a high-priority request, and not just a disturbance. After the interrupt we assume that the high-priority message is transferred, i.e., HPM in Figure 4.2, and finally the normal TDMA communication resumes, using one of the approaches Alt. (a, b, or c), shown in Figure 4.2.

An example of a protocol using a similar method is the QWIK protocol [kSkJC01, kSkJC01]. The QWIK protocol starts and restarts its TDMA-round with a reset-pulse. However, if a reset pulse is sent, the schedule immediately starts from the beginning. This means that if we want to interrupt a TDMA-round for a high-priority message, this message must already be scheduled in the TDMA-round.

The reset method increases the complexity of the communication protocol, and may have reliability implications, e.g., some of the deterministic behavior is lost and the times when a node accesses the bus is no longer fixed.

Using this method we must design the system such that a reset pulse is used very infrequently, otherwise we have to consider collisions when starting to send after the reset pulse. However, in most cases we would still need some media access method to ensure that two high-priority messages do not collide. One of the alternatives to avoid collisions would be

minislotting, see Section 1.2.1, our use of this approach will be described in the next section.

### 4.2.2 Minislots

In this approach we use minislots to arbitrate among nodes that are allowed to send higher priority messages. In Figure 4.3, we show how message-frames are separated with a number of minislots, i.e., short slots of media silences. Thus, after each periodic message a minislotting period is used to make it possible for high-priority messages to be sent. The length of this minislotting period, i.e., the number of minislots, depends on the number of nodes that are allowed to send high-priority messages. In Figure 4.3, we can also see how a node with a high-priority message starts to send its high-priority message, after it is assigned the minislot period.

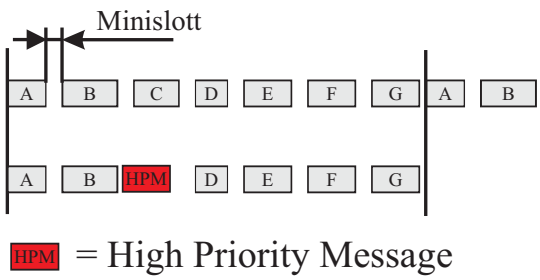


Figure 4.3: Interrupt the TDMA-round by minislots between, pre-scheduled messages.

### 4.2.3 One Minislot

In this approach we use only one minislot, i.e., a slot of the length of the channel end-to-end propagation delay. In this slot the transfer of a high-priority message can be initiated. We assume that these high priority messages occur rarely, otherwise they should be scheduled as normal messages in the normal TDMA schedule. The probability for collision among these high-priority messages should be low. However, if a collision occurs the following steps will be used. Note that the colliding nodes are not aware of the identity of the other node(s) in the collision.

1. The colliding nodes withdraw.

2. Higher priority messages are divided in two sets according to their priorities. For example, assume 1 is the highest priority followed

by priority 2 etc. A message with priority $i$ is denoted $m_i$. Assume we have 9 high-priority messages and the detected collision occurred between messages $m_x$ and $m_y$, where $m_x$ has the higher priority ($x < y$). Messages are divided into two sets according to their priority, i.e., 1,2,3,4 and 5,6,7,8,9. Note that this is a distributed activity where the division-behavior is static and decided pre-runtime.

3. We now allow the high-priority set to send immediately after the collision.

4. If only $m_x$ was in this set, no collision will occur and the highest priority message is sent.

5. If both $m_x$ and $m_y$ are sent and cause yet another collision, the high-priority set is divided into two new sets 1,2 and 3,4 and we return to step 3.

6. If none of $m_x$ or $m_y$ are sent, there will be silence on the media for one minislot, and we know that $m_x$ and $m_y$ are in the lower priority set. This set is divided into 5,6 and 7,8,9 and we we go back to step 3.

When message $m_x$ has been sent, the procedure will be repeated for message $m_y$. This can be handled in two ways, (a) retry sending $m_y$ directly after $m_x$ or (b) after one message from the normal TDMA-schedule. In both these cases, the minislotting behavior must be repeated to avoid collisions with newly arrived high-priority messages. Method (a) is naturally faster which is important for high priority messages. However, the synchronization aspect could benefit from using method (b), as synchronization is normally based on the periodic arrival of pre-scheduled messages.

## 4.3 TTP+: An New Approach based on TTP/C

In this section we present a new approach for sending event-triggered data on a time-triggered channel. We develop a modified version of the existing TTP/C protocol [TTP99, KG94] as the base protocol. Our modification is labeled as TTP+. Our intention is to permit as few changes as possible to the TTP/C protocol so as to maintain its predictability feature, and also not to impose any additional complexity. As stated in the previous section, there will be no "preemption" of the normal time-triggered communication in order to send higher priority messages. Instead we have

restricted this case to low-priority event-triggered channels, where the focus is on robustness and short access times.

To facilitate sporadic data transfer with TTP/C, our approach introduces two new concepts, (1) the Sporadic Information Transfer (SIT) bits, and (2) *free-frames*. The SIT bits consist of one or more bits, included in the normal data frames. The purpose with the SIT bits is to either directly send sporadic data or requesting some additional slots for sporadic data. After such a request a node will normally be assigned a slot, i.e., a *free-frame* in which it can send its sporadic data. In Figure 4.4 a normal communication frame is shown with an additional SIT bit. This normal data frame makes it possible to implement the required functionality without any major changes.

Conceptually, the *free-frames* are empty slots reserved for sporadic/event-triggered data. These newly introduced *free-frames* are part of the communication schedule, as they are statically scheduled. However, any node is allowed to send sporadic data in these *free-frames*, see Figure 4.5. This will require a method for avoiding collisions in these slots, and we will come back to how this is handled in Section 4.3.1.

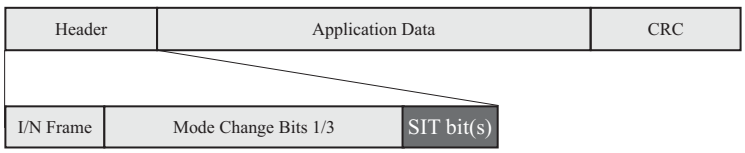


Figure 4.4: A TTP/C normal frame with the addition of a SIT bit.

The *free-frames* are used when considerable amounts of sporadic data is needed. In such cases, the introduced SIT bits will be used as indicators that a node wants to use the "*free-frames*". If data is sent directly using the SIT bits we get a low bandwidth solution and, using the *free-frames*, we get a higher bandwidth solution. Thus, depending on the amount of sporadic data a node wants to transfer, we use the SIT bits in two different ways: (1) in case of small amount of sporadic data, the SITs bits are used to transfer data, or (2) in case of larger amounts, the SIT bit can work as requests for *free-frames*.

The decision to use only a single or several (yet very few) SIT bits is based on limiting the overhead for introducing a sporadic channel. However, using only a single bit has some implications, e.g., we have to decide how to use this SIT bit pre-runtime. When transferring data via the SIT bits, we call it a Low Bandwidth (LB) solution. The available bandwidth for sporadic messages at run time is static in such a case. If

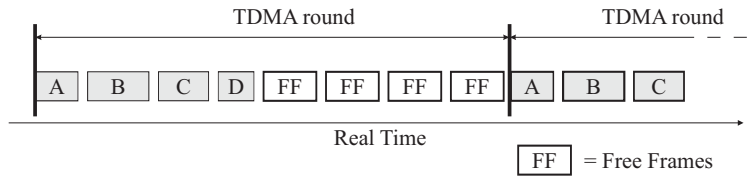


Figure 4.5: A TDMA-round with event-triggered channels in form of *free-frames* located in the end of the TDMA-round.

a node uses LB, it needs a local priority queue for sporadic messages. In case of more than one receiver we have to handle addressing, start and stop of messages, etc.

By using the SIT bits for message transfer we have created a low bandwidth channel for event-triggered communication over a time triggered system. The advantage of this method is the low complexity which facilitates the design of a robust system. However, it offers limited bandwidth and that bandwidth can not be used by any other node when nothing is sent.

In the second option, we used the SIT bits as requests for further sporadic bandwidth, i.e., requests for *free-frames*. This makes it possible to share the *free-frames* among all nodes, i.e., all nodes can use this bandwidth. This approach implements a High Bandwidth (HB) event-triggered channel for nodes that have a lot of sporadic data to send. This also has implications on the working of the protocol. Specifically, we have to decide how the sporadic bandwidth is divided among requesting nodes. There is a number of options which we will discuss in Section 4.3.1. Finally, a node can use a combination of the above such that the node has a LB channel and a HB channel.

In the following sections we will describe and investigate how a number of variants on assigning *free-frames* affect the event-triggered channel. Another parameter that is investigated is when the *free-frames* are scheduled, e.g., composed at the end of a TDMA-round or scattered over the whole TDMA-round.

### 4.3.1 Prioritization of Sporadic Messages

In this section we discuss alternate solutions for controlling the *free-frame* access among nodes; and also across messages within a node. The goal is to obtain short access times and high throughput for the sporadic data.

We only consider how nodes will share the *free-frames*, as communication via the SIT bits is strictly handled node internally. Locally, a

node must handle sporadic messages such that they get queued in a node internal queue according to their priority. This is independent of whether the data will be sent via the SIT bits or *free-frames*. To transfer sporadic messages we need to handle extra information, compared to the periodic data, as the sporadic traffic is not static:

- Start and stop information of sporadic data.
  - Start and stop bits, indicating the start and stop of a message.
  - Messages can be sent starting with a specified offset from the start of, for example, the cluster cycle, i.e., a number of repeated TDMA-rounds, or TDMA-round. When using sporadic messages with predefined length, they can be synchronized to the clusters cycles.
- Destination address or message ID serving as address. This is necessary to transfer in order for receivers to know two whom the message is directed. However, sender address is not necessary as the node from which the frame arrived is known.

As discussed earlier, the *free-frames* approach requires a media access method to avoid collisions when sending (using) the free frames. We have investigated two approaches for avoiding collisions, one central and one distributed approach, described in the next two sections.

### Central Prioritization

In this approach, one node will make a central/global decision about which node is allowed to send in a specific *free-frame*. The nodes sending in the static area can request bandwidth for sporadic data in the form of *free-frames*. The last node in the static part of the TDMA-round, i.e., node $i_e$, will prioritize and decide which of the requesting nodes are allowed to send, and at what time. Node $i_e$ will then include the schedule of the sporadic event-triggered data in its message, see Figure 4.6.

Using this method, we get extra overhead for explicitly sending the schedule for the event-triggered traffic. If we assume that we indicate whether a node should send or not, we use $n$ bits followed by $n$ times the maximum number of *free-frames* that are sent per node, assuming $x$ bits we get a total overhead (H) of:

$$H = n + n \cdot x$$

One consequence of using this approach is that the schedule is explicitly sent on the bus. It is a low complexity solution, but only one node

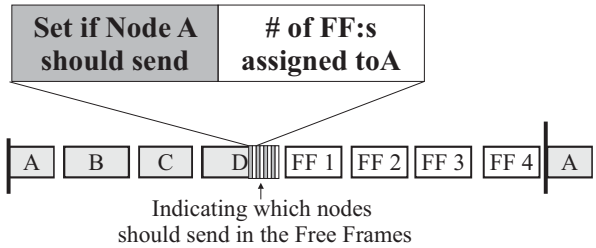


Figure 4.6: A central node decides which nodes are allowed to send in the event-triggered channel, formed by the *free-frames*.

controls the schedule, which can potentially introduce a single point of failure. However, normally the safety-critical information is transferred via the more predictable time-triggered channels. Only when using one central unit for prioritization of the event-triggered messages can the implementation be easily changed without having to make changes at all nodes. Nodes that do not receive this message are considered to observe silence semantics.

### Distributed Prioritization

The previous method has the disadvantage of introducing a single point of failure (if the central node fails the event-triggered channel will collapse). A more attractive solution, with fault tolerance aspects, is to use a distributed approach when deciding the allocation of *free-frames* to nodes. In this solution, each participating node makes a decision based on received information, which makes it very important that all nodes receive the same information. Thus, the nodes make a distributed decision about which nodes can access the *free-frames*. Our method has the purpose of achieving low overhead, small delay, and uniform bandwidth among requesting nodes. However, this basic method can easily be modified to give one node higher priority. Although, in the following we only describe the basic method where all nodes will be able to send and where we focus on short media access times.

The sporadic transfer efficiency is dependent on the number and positions of the *free-frames* in the TDMA-rounds. This approach basically gives priority to the node with the earliest request. For example, assume nodes $a$, $b$, and $c$ send (in that order) their time-triggered frames, just before a *free-frame* is scheduled. If all request a *free-frame*, then node $a$ will get the highest priority followed by $b$ and then $c$. If there were other nodes in the queue before $a$, $b$, and $c$ that could make there requests, they

will be put in the queue after the new nodes, i.e., node $a$, $b$, and $c$. This means that nodes might not get access to the event-triggered channel if there are more nodes than *free-frames* in the system.

We also note that in order to minimize the access time to the event-triggered channel, there should be a *free-frame* in every second frame, see Figure 4.7.

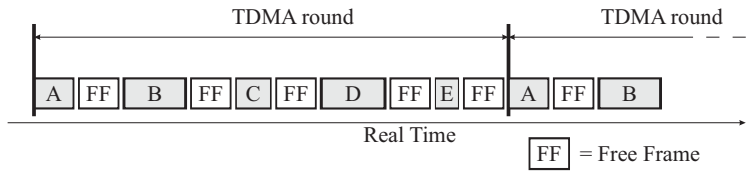


Figure 4.7: Distributed method where the *free-frames* are scattered throughout the TDMA-round in order to minimize the buss access time for sporadic data.

If a node fails to receive a message with a request for a *free-frame* it will get an inconsistent view of the global *free-frame* queue. In such cases there must be a mechanism that makes it possible for a node to regain this global queue information, such that a node can reintegrate in the transfer of sporadic messages via *free-frames*. This is done by limiting the size of the global queue and enforcing nodes to re-queue their requests, each TDMA-round. Thus, when it is a node's turn to send, say node $a$'s, $a$ is removed from the queue, independently on whether $a$ was assigned a *free-frame* or not. If node $a$ still has sporadic data to send in the local queue, it must request a *free-frame* again. Thus, after one TDMA-round the global queue is renewed.

The event-triggered channel works by a combination of the global queue and a node's local queue which is strictly priority based. Thus, a sporadic message $m$ is put in the local queue of a node $a$, according to the message priority. Node $a$ requests a *free-frame* to send this message in. This request is queued in the distributed global queue and $a$ is assigned a *free-frame* accordingly.

### Optimizations

The method of queuing requesting nodes described above can easily be adapted to a system using global priorities. This can be achieved if each node in addition to the *free-frame* request includes the priority of the message. Then nodes can use the priority when queuing the requests in the distributed global queue. This will naturally cost more in overhead

as the priority of a requesting node's message is transferred on the media as well. However, in this type of system, where a lot of parameters can be set before runtime, there is a possibility to minimize the number of priorities in the system. This can minimize the number of bits necessary to transfer the requests and corresponding priority.

When half the frames are *free-frames*, each node will have the possibility to send in a *free-frame* each TDMA-round. If there are less *free-frames* and all nodes want to send, the lowest prioritized nodes will not have the chance to send. Thus, the will be subjects to starvation if they never can access any *free-frames*. However, this could be handled by circulating the priorities among nodes during different TDMA-rounds.

During times when no nodes request event-triggered data transfer, the *free-frames* can be statically assigned to specific nodes. This serves two purposes:

- A nodes' response times can be reduced by assigning default designations to *free-frames*. Thus, all *free-frames* will be preassigned to a node. If a node notices that its designated *free-frame* is not reserved, i.e., requested, it is free to use that *free-frame*. If this node has data to transmit, and did not have the opportunity to request a *free-frame*, it is free to send in that *free-frame*. This can, during low load situations, lower the access time for nodes.

- Nodes that normally do not send every TDMA-round, can be assigned a *free-frame* which gives such nodes access to those TDMA rounds under low load situations, i.e., those nodes have the possibility to send in rounds when they are normally not sending any time-triggered frames.

With the described method, a node may be assigned more than one *free-frame*. However, if one *free-frame* suffices for the node, it would occupy more *free-frames* than necessary. As we use the same message-frames both for time-triggered and event-triggered messages, we can use the SIT bits to indicate when a node has no more sporadic data to send. Thus, if a node is assigned two or more *free-frames* but only needs one, it can indicate "no more sporadic data" using the SIT bits in the *free-frame*. This will improve the utilization of the media, as no *free-frame* is assigned to a node with no more data to send.

## 4.4 Properties

In this section we briefly describe some of the properties of our adaptation of the existing TTP/C protocol and some preliminary simulation results

where the intent is chiefly to confirm that the TTP/C behavior has not been perturbed. We defer detailed simulations as a future refinement to the approach. We have focused on our approach using distributed prioritization, from Section 4.3.1, where every second slot is a *free-frame*, see Figure 4.7. We have also assumed that all time-triggered message-frames have the same length, and all *free-frames* have the same length. However, time-triggered message-frames and *free-frames* do not necessarily have the same length.

We emphasize that our main focus has been to achieve flexible handling of sporadic messages without disturbing the time-triggered base. In Table 4.1 we compare a few important properties of our modified TTP-protocol, termed as TTP+, with the classic TTP and the CAN [CAN91] protocols. For the TTP protocol we assume that sporadic messages are handled using preassigned slack as described in Section 4.1.1. In Table 4.1 we show the (1) worst-case (WC) delay of a sporadic-message, (2) the overhead corresponding to sporadic message handling, and (3) how much sporadic data can be assigned to a single node, under the condition that they all have the same amount of data and the same period $T$ of periodic messages.

In the first column we have the WC delay where our TTP+ and TTP have the same WC delay, i.e., one period T. The CAN protocol has a very short WC delay, which is when the longest message must finish sending before the next may access the bus.

In the Overhead-column of Table 4.1, the overhead related to the sporadic message transfer is shown. In our TTP+ the overhead is related to the SIT bits, one for each node, assuming $n$ nodes. For TTP using pre-assigned slack we have no extra overhead. For the CAN protocol, the ID-field is used to resolve priorities accessing the media, and thus it has more functionality than just sporadic messages. Thus, it should be noted that comparing these may be a bit favorable for the TTP protocols.

Finally, the last column indicates how much media access can be assigned a single node, in the best case. For our TTP+ we can basically assign all *free-frames* to one single node, i.e., $T/2$. For the standard TTP we cannot change the pre-scheduled slack, and assuming each node is assigned the same amount of slack for sporadic messages, one node gets the size of approximately one *free-frame* ($FF$). In this column we can see the major improvement of our TTP+ approach which allows a node to utilize more than one *free-frame*. This significantly improves the transfer rate, when a single node has a lot of data to transfer.

One question naturally arises as to how the average delays are affected. In the following simulation our goal has been to present some preliminary validations of the expected behavior of the TTP+. In future

| Comm. approach | WC delay | Overhead | Max trans |
|---|---|---|---|
| TTP+ | T | n bits | T/2 |
| TTP | T | 0 bits | approx. 1 *FF* |
| CAN | Max ML | n · ID-field | T/2 |

Table 4.1: Properties of different communication approaches

work, we plan to conduct more detailed simulations using varying parameters of TTP+ and simulation parameters. In Figure 4.8, we observe that although we have introduced a more flexible handling of sporadic messages in the TTP protocol with TTP+, we still maintain, at least, the same average delay of sporadic messages. At low load we even manage to improve the average delay, although the only optimization, described in Section 4.3, implemented is simply that the *free-frames* have preassigned nodes. We believe that refined optimizations will help improve the average delay characteristics even further. In Figure 4.9, we have verified that this behavior is still valid if we increase or decrease the size of the *free-frames* compared to the normal time-triggered message-frames.

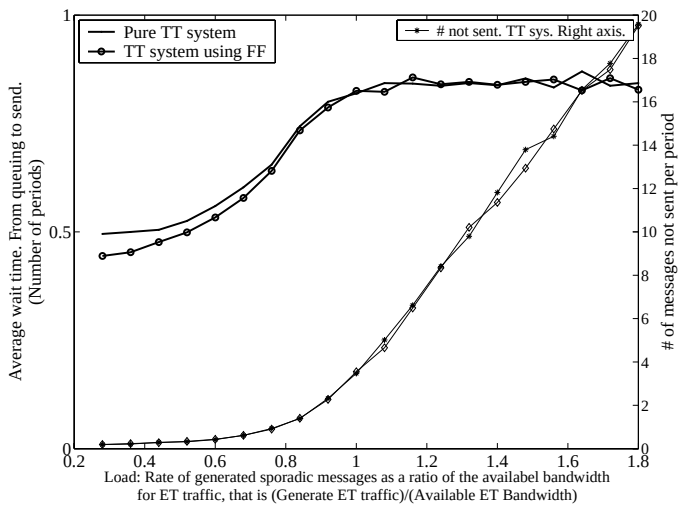


Figure 4.8: A comparison of our TTP+ and a basic TTP using preassigned slack.

## 4.5 Summary

In this chapter we have studied and presented different approaches for transferring sporadic messages using a communication system based on

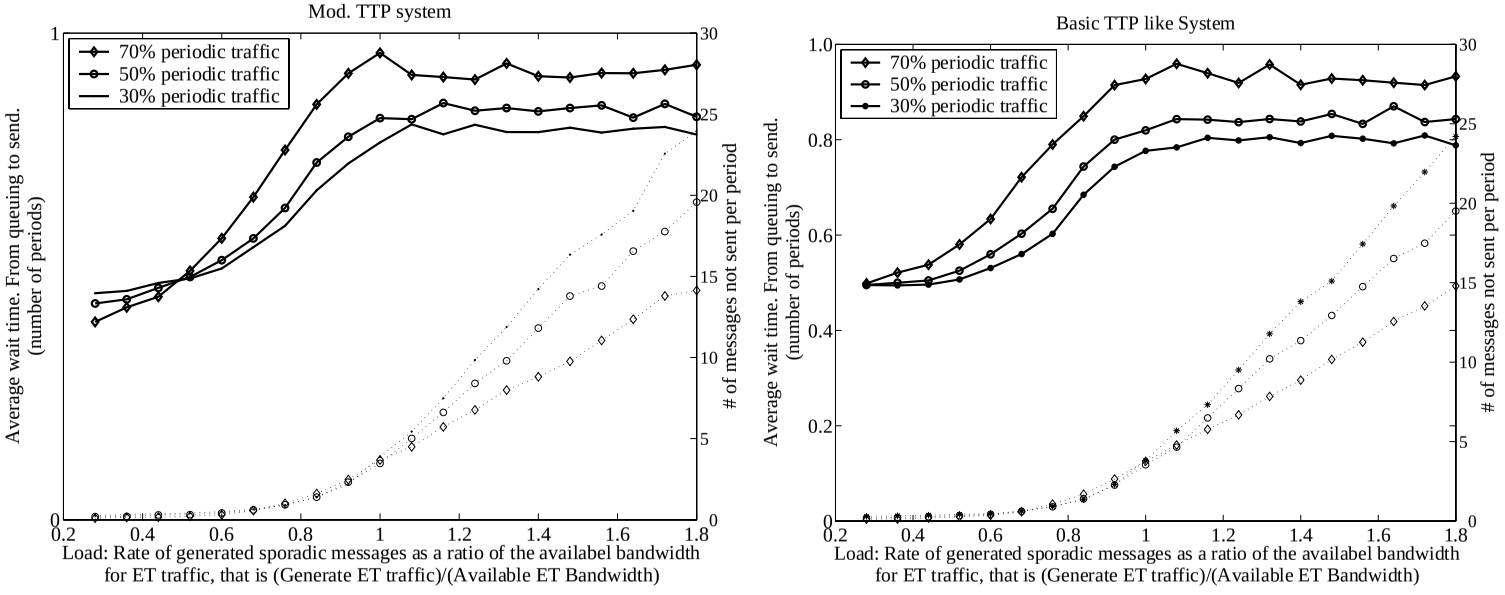


Figure 4.9: These figures shows the basic TTP and the TTP+ have very similar, behavior when changing the amount of the time-triggered part.

the time-triggered paradigm. We have studied both how priorities higher and lower than the normal time-triggered traffic can be handled. It is, however, clear that transferring high-priority messages by interrupting the normal time-triggered communication has a significant impact on the a time-triggered protocol. Naturally, it is more difficult to maintain the reliability aspects as it affects the predictability of the time-triggered communication paradigm.

For low-priority sporadic messages the situation is a bit different. We have shown some basic methods of how such messages can be transferred using a time-triggered base. And more importantly, we have presented a new method based on the TTP/C protocol for transferring sporadic messages in a more flexible way. Our main focus has been to provide for enhanced flexibility in communication without disrupting the fundamental predictability of the TTP/C protocol. This has been achieved with a minor change in normal message-frames. Using our new approach, nodes share *free-frames* on a request basis. This is a significant improvement, compared to using preassigned slack, as one node can utilize many *free-frames* when another node has no sporadic messages to send. Our preliminary simulation results show that, without disturbing our time-triggered traffic, we have maintained the average delay times compared to a time-triggered approach using pre-assigned slack. At low load we have even improved the average access times. We have also shown with the preliminary simulations that there is no indication that these results are affected by the size of the *free-frames* compared to the normal time-triggered frames.

# Chapter 5

# Conclusions, Perspectives and Future Issues

THIS chapter summarizes the main results achieved in this thesis. Alongside, we mention future research areas that we foresee from the perspective of the work conducted here.

This thesis has concentrated on the development and investigation of communication approaches in the area of multiple-access media. The assumed working environment for these communication systems is distributed embedded systems with reliability requirements. We expect such computing systems to rapidly grow within the mass-market arena, where efficiency and cost are salient factors alongside performance and reliability. This has motivated us to use efficiency and cost as the main design drivers. In the following sections, more specific details of the contributions of this thesis are presented.

## 5.1 Synchronization Issues

In Chapter 2, we presented a unique synchronization approach that provides for low-cost fault-tolerant synchronization in distributed real-time systems, targeting safety-critical systems for the mass-market. The proposed TDMA (Time-Division Multiple Access) communication approach uses the static information of the different message lengths to obtain information about sender **id**. This simple approach provides for low complexity and high efficiency start-up synchronization and subsequent re-synchronization to integrate new/recovering nodes. Fundamentally, our approach makes the following contributions:

- Provision of guaranteed and time-bounded start-up synchronization.

- Short average start-up synchronization bound, with small standard deviation.
- Low communication overhead compared to existing TDMA-based approaches.
- Fast re-integration of recovering nodes.
- High robustness in tolerating faults with only a limited synchronization and resynchronization time penalty.

Thus, the synchronization approach appears well suited for real-time systems, due to its guaranteed temporal upper bound on start-up. Its low complexity, combined with its simplicity, makes it useful for cost-sensitive safety-critical systems as well.

## 5.2 Media Access

In Chapter 3, we have studied two contemporary media-access methods, the time-triggered approach and the event-triggered approaches. We have compared these two methods to assess their behavior under different operational conditions. This chapter intends to help system designers determine the range of behavioral differences these two approaches provide in varied scenarios.

The work in this chapter has been conducted using two types of simulations where we have studied the behavior of these approaches. The first simulation type used a system with a mix of tasks and communication attributes. In the second type, we studied the system by looking exclusively at the communication media. The intent is not to re-establish the basic properties of these communication approaches, as they are well-known in the community. Instead, the focus has been on quantifying attributes, as well as the quantitative spread of differences across the approaches, and also under which conditions their behaviors best matches the system and communication requirements. Considering the trade-offs during system design, it is important to know the relative gains of choosing one approach over another, and also ascertaining the operational profiles where each approach fits best. In this work, we clarified some communication protocol aspects. Naturally, a system designer must take into consideration other aspects of these approaches, for example, the predictability of the time-triggered approach. Other facets target issues such as scalability of the approach to varied system sizes, along with issues such as stability and design aspects like composability.

Here we describe the main conclusions of the work of Chapter 3, under the operation conditions described there.

The **Event-Triggered Architecture** (ETA) is beneficial when short delays are important and the load is kept below approximately 0.6–0.7. As detailed in Chapter 3.4, the load is defined as:

$$Load = \frac{Generated\ sporadic\ traffic}{Available\ sporadic\ bandwidth}$$

The main properties supporting the ETA include:

+ This approach provides for short average delays for messages that access the media. Even when the load increases above (approx. $>$ 0.6), there is only a slight increase in the average delay for the high-priority messages that get access to the media.

– The use of event-triggered communication unfortunately also delays high-priority (periodic) messages. These delays originate from the fact that sending messages cannot be interrupted, thus they also cause varying delays, i.e., jitter. High-load situations may lead to starvation of low priority messages, i.e., there are messages which are prohibited from accessing the media due to higher-priority messages constantly accessing the media. The variation in delays grows early around a load of 0.5.

Comparatively, the **Time-Triggered Architecture** (TTA) supports the system requirements sufficiently for all tested loads, as long as the requirements on delays for sporadic messages are not very high. In such a case, the TTA is preferable only for loads above approximately 0.6–0.7. The main properties of the TTA are:

+ Naturally, and perhaps the most important fact to highlight about the TTA is that there are no delays of high priority (periodic) messages. When most deadlines for sporadic messages provide for two or more tries, i.e., a delay in the order of a few TDMA-rounds, there is actually few missed sporadic messages. In such cases, it even creates an equalizing effect providing for more low priority messages, naturally at a cost of higher average delays.

– However, when we have short deadlines for sporadic messages, most sporadic messages only get one chance to send a message. This corresponds to situations where most deadlines have slightly more than a TDMA-round to send, see Figure 3.2. The average delay starts with as much as a half TDMA-round, and increases slightly as the load increases.

It is important to re-emphasize that the terms "flexibility" and "predictability" of existing event-triggered and time-triggered schemes are subjective. There is a tendency to demonstrate the strength of their features in system models and operational situations that tend to favor the stated approach. Thus, the intent of our work has been to choose a common system and communication model, and conformal definitions of load, as we quantify the properties of each approach. As we mentioned earlier in Chapter 3, these results are based on simulations with randomly generated loads. It could be argued that more "real" loads should be used. However, considering the lack of publically available and published task schedules from industry using those as a base could easily benefit one or the other approach. We have therefore chosen the approach of using random generated communication loads such that there would be no doubt that they favor none of the approaches.

It should be pointed out that these results are not directly adaptable to real-life situations. The intent is that the basic properties of the generated load can be found in many real-time systems. For example, systems with high-priority control loops generating high-priority periodic messages and other system parts generating sporadic messages, e.g., with diagnostics and status information can be found in many application areas.

Therefore, we believe that our quantification provides a system designer with meaningful profiles for both the approaches. Of course, the designer will make the changes specific to the actual system requirements driving any development - profiles such as the ones developed in Chapter 3, help the designer to understand operational regions where each technique offers strengths.

## 5.3 The Best of Both Worlds

In Chapter 4 our goal was to develop methods for combining the favorable properties of predictability and flexibility, respectively, of the time-triggered and event-triggered communication.

As our focus is on real-time and reliable embedded systems, we consider the predictability of the time-triggered approach as a key factor for the provision of safety-critical services. Hence, the time-triggered approach is used as a base, upon which we present new methods as well as modifications of existing methods for efficient handling of event-triggered traffic and sporadic message handling. We emphasized in Chapter 4 that although we desire composite time- and event-triggered handling, a primary constraint we impose for ourselves is to minimize any disruption of the predictability established by the time-triggered base.

Overall, we have presented three main approaches for the integration of event-triggered and time-triggered communication:

1. In the first type of integration, we present a number of fundamental approaches for transferring low-priority sporadic messages via a time-triggered channel. With low-priority, we mean lower priority than the regular periodic messages.

2. Here we consider high-priority sporadic messages. This has a higher impact on the regular time-triggered traffic, as we must interrupt the TDMA-round to achieve better message delivery times.

3. The final step in this work has been to improve the sporadic message handling of an existing communication protocol, the TTP/C protocol [TTP99]. This has been done with very limited changes to the original protocol specification. Thus, there is no disruption of the normal time-triggered traffic and minimalistic specification changes in order to not disrupt the behavior of an already well-verified and tested protocol. It is worth pointing out that our proposed changes do not change any operational specification of the TTP/C protocol that is required over its property verification.

## 5.4 Future Research Issues

Following the presentation of the main contributions of this thesis, we discuss some future research issues that are of interests. We have structured these along the same topic lines as used in the thesis.

### 5.4.1 Synchronization Issues

The term "synchronization" in distributed systems has varied connotations and also spans a very wide area, see [SHW94]. The restrictions imposed by commercial embedded systems - cost issues and physical distribution of nodes - have made bus-based communication the significant media on which to develop synchronization and other services.

In our work, we have focused on providing cost-effective and efficient initial synchronization for TDMA-communication. At the same time, synchronization is a basic system service on which more complex services, such as group membership, etc., are developed.

- What are the scalability issues for TDMA-based techniques? In Chapter 2.6.3 we demonstrated that our approach with unique message-length identifiers scales even when a large number of nodes

is used. However, the effect was a slight increase in start-up time. Is it possible to use other techniques to enhance the scalability without degradation in average and bounded start-up times? Furthermore, is the fault model of a node that ends up mimicking the unique message lengths of another node a concern? We have considered this as a minimal risk as messages are protected with checksums which will limit this considerably. However, a more thorough investigation would be interesting, especially as we extend our approach to consider safety-critical areas. Finally, it would be interesting to investigate for which bit encoding approach this synchronization approach would fit best, e.g., Manchester encoding or NRZ.

- The work within this thesis has focused on synchronization and media access methods, especially, the time-triggered and event-triggered paradigms. However, systems can be divided according to the synchrony among constituting components and their perception of time. As one extreme we have systems with no explicit notion of time, i.e., asynchronous systems. As the other extreme we have synchronous systems where nodes utilize an explicit consideration of time for event ordering, e.g., using local/global clocks, and a bounded time on executing a step. In synchronous systems, messages are also received within a bounded time window. The system nodes may furthermore be synchronized with each other.

  Independent from the media access method, a system designer can choose to use a synchronous or an asynchronous system approach. However, using a strict TDMA system already requires a common view of global time amongst the nodes of the system. This enforces some minimum synchrony amongst the nodes and the system is not strictly asynchronous. Although it might seem natural to take advantage of this synchrony when implementing the distributed applications, there might be reasons for not doing so. For example, the system designer may want to reuse code or make the code independent of the underlying communication system.

  Other media access methods generally use little synchrony information in their manner of operation. This might seem a more natural approach if one is using an asynchronous system model. However, nothing prevents implementation of additional synchrony in the distributed media access protocols

  In this area we would like to further investigate these more general concepts of system time, and expand on their implications of the time-triggered approach and the event-triggered approach.

### 5.4.2 Media Access

In Chapter 3, we investigated the relative behavior of two well-known media access approaches, i.e., the time-triggered approach and the event-triggered approach. Although it serves the purpose of showing a system designer their behavior under similar working conditions; however, there are a few directions in which we would like to enlarge our investigation.

We would further like to run experiments where the load includes situations where we media bursts, i.e., gets overloaded. The intention is to verify results from our current investigation, where the load is randomly generated but more evenly distributed throughout the run. We have conducted different runs with a wide range of loads, which we believe also covers bursty situations as we have run simulations with very high loads. However, it would still be interesting to see whether this is confirmed or if the fact that alternating between short periods of very high load and of very low load favors one or the other approach.

The work in Chapter 3 is naturally closely related to Chapter 4, where we study more efficient solutions of transferring sporadic data using a time-triggered base. It will be a natural extension to the work done in both Chapter 3 and Chapter 4 to extend these simulations with our new approaches for sporadic message handling to evaluate there behavior. Furthermore, we can investigate the limits of adding event-triggered traffic, and are there any circumstances when one or the other will start suffering?

### 5.4.3 Composite Event-Triggered and Time-Triggered Approach

The work of integrating event-triggered communication over a time-triggered channel has resulted in some new and interesting approaches. However, although these approaches look promising, some further investigations are necessary. For example: Is there any issue of ET-specific and TT-specific fault modes that may be of concern? Can we utilize any specific architectural support, e.g., from the time-triggered base, to ensure separation of event-triggered and time-triggered message handling, i.e., separating non-safety-critical and safety-critical messages? Similarly, can we use specific architectural support to increase the efficiency of sporadic message handling?

Using active star couplers is one proposed way to overcome the difficulty of achieving a broadcast channel with fiber-optics, especially implementing a fiber-optic bus. This also provides us with a central way of ensuring that no node, can monopolize the bus due to a fault, i.e., we can avoid babbling idiots. Furthermore, this also provides us with an

additional way of arbitrating the channel. This can be done by putting logic in the active star coupler such that it can decide which node is the next to be allowed to send, but it could also detect high-priority messages and give them priority. However, this would naturally change the system environment. This might give new possibilities for how to efficiently integrate event-triggered and time-triggered communication, but can also impose limitations.

# Appendix A

# Appendix A

## Proposal for a distributed computer control system architecture in heavy duty trucks

**Vilgot Claesson**
Department of Computer Engineering, Chalmers University
**Magnus Gäfvert**
Department of Automatic Control, Lund University
**Martin Sanfridson**
Mechatronics Lab, Royal Institute of Technology

Technical report no. 00-16
Department of Computer Engineering
Chalmers University of Technology

The aim of this work is to perform a design of a safety critical distributed control system by applying and combining methods within the areas of dependable computer systems and control theory. Many such systems with interesting requirements and constraints regarding performance, dependability, and cost, etc. are found in the area of automotive engineering. We have chosen to work with a brake system on a heavy truck-trailer combination vehicle, with certain additional functionality. This system is a distributed safety-critical control system by nature, and constitutes an excellent case study for this purpose.

It is our belief that there is a great potential in combining theory from the areas of computer engineering and control theory in the construction and design of dependable distributed control systems. There exist many methods and paradigms in computer engineering on how to build dependable distributed computer systems. The results are often independent of the application. It is assumed that all failure handling and avoidance is taken care of by the computer system. It is assumed that failure to fulfill the requirements at any instance leads to system failure. If the computer system hosts an implementation of a control system, this is not necessarily true. It may be possible for the control algorithms to take care of a failure (transient or permanent) in the computer system, and continue to run the system in a safe, possibly restricted, mode. This opens the door for possibilities to relax the requirements on the computer system, thus making it possible to decrease the complexity and the cost. In control theory, on the other hand, there are many results that deals with design of robust distributed control systems. In this context the underlying hardware; the computer system and the network, is often treated in a highly abstract manner. It may be regarded as the source of stochastic network delays, a possibility of lost samples, a finite resource for computation time or bandwidth, etc. In a certain application it is necessary to know the properties of the computer system with respect to these and possibly other characteristics. The computer system is often taken for granted, in the sense that is it regarded as a fixed prerequisite in the control system design.

The motivation for this work thus is the possibility of better and more effective system designs if the combination of the computer system and the control system is regarded at an early stage. This can be reached by ensuring that the computer system has certain properties from the control engineers point of view, and vice versa, by ensuring that the control system has certain properties from the computer engineers point of view. Therefore it is necessary to specify the requirements of these systems with respect to each other in a detailed and systematic way. This can, for example, include what services the computer system and communications

network should provide, what bandwidth requirements they should fulfill, etc. For the control system this may be specifications that the control algorithms should be able to handle network delays of certain characteristics, transient failures such as lost sensor data for a certain time period, mode switches to safe limp-home modes, etc. To successfully carry out codesign of the computer system, the communications network, and the control system it is necessary to specify interfaces between these systems in the design process and in the implementation. In the design process these interfaces consist of requirements and specifications, and in the implementation they may consist of certain functionalities, such as software services or functions.

In this work specific focus is placed on the choice of distribution level of the computer system, i.e., the computational power, and the control system. The distribution levels of these systems are orthogonal in the sense that they can be chosen independently for each system, and then be combined to a complete system. Though in practice not all combinations are useful. The combination is performed by mapping specific control system architecture on a computer and communications architecture. To explore these combinations we regard three different distribution levels on the control system and computer and communications system, respectively. These combinations will be investigated regarding dependability and required information exchange. The dependability of the computer architecture will be investigated both with and without redundancy. Furthermore, a short discussion of the effects of fault will be presented.

The distribution levels of the control and computer architectures will be presented in the following sections. The combinations of these will be analyzed and discussed, resulting in a proposal on a system architecture with respect to distribution levels.

## A.1 Acknowledgments

We want to express our gratitude towards Mats Andersson, at Volvo Technological Development for his support during our visits to Volvo. We would also like to include Olle Bridal, Jacob Axelsson and Olof Lindgrde, from the same company, in our gratitude for fruitful discussions. Furthermore, we want to thank our supervisors, Bjrn Wittenmark, Neeraj Suri, and especially Martin Trngren, for helpful comments. The project is financed by NUTEK under project P11762-2.

## A.2 The Target System for the Case Study

The target system in the case study is a truck and semi-trailer combination. The functionality of this case study implementation is the control system for normal braking, ABS, and yaw-control, for abbreviations see Appendix B.1. The purpose of yaw control is to prevent rollover and skidding. For this system an appropriate distributed computer architecture will be proposed and the co-design of control and distributed computer system will be investigated. The communication system of the truck will be investigated, however, the communication network of the semi-trailer will not be treated. To be more detailed, the distributed functionalities involved are: ABS, TRC, AYC, and ARC. A common name for these is VDC (Vehicle Dynamic Control), which is implemented in an Electronic stability program (ESP). As far as possible the environment of the VDC functions will be included in architectural considerations, for example, functions that interface this control function.

The dynamics of the vehicle combination is treated in [GSC00].

## A.3 System Constraints

The system is built upon a bidirectional broadcast bus. To the bus one or more computer nodes are connected. If only one node is connected, we consider it to be a central computer architecture, sometimes referred to as a distributed I/O system. Sensors and actuators are in this case connected directly to the bus. Two types of system will be studied: a simplex system and a redundant system. In the redundant system critical functionality will be duplicated, and we will refer to it as a duplex system. The basic requirement of the critical parts of the system is to tolerate a single component failure. However, due to very hard requirements on low cost we assume that only limited redundancy are possible. Therefore, our study has been limited to duplicated components, and solutions using more redundancy, such as Triple Modular Redundancy, have not been consider for cost reasons. Using a duplicated system voting is not possible, we will instead assume that nodes are fail-silent, i.e., they produce the correct result or no result at all.

When replication is used the replication strategy is active redundancy. This means that both components of a replica run at the same time and consume and produce the same data. Active redundancy is used since it is assumed that the startup time for initiating a new unit is too long.

No assumptions are done about the communication protocol except that it uses a broadcast bus.

## A.4 Architecture

This project strives to give a good interface between the control applications and the computer architecture, in such a way that they fulfil their mutual requirement on each other. Therefore, the architecture of the control application and computer system will be discussed in this section. It is important to choose a computer architecture that supports the control applications in the best possible way. However, it is also beneficial if the control architecture can be chosen in such a way that the complexity and cost of the computer architecture can be reduced.

To reduce the problem we have chosen three different approaches of both computer and control architecture. These different approaches will be combined into nine cases. The reliability aspects of the computer architectures will be then be investigated. Furthermore, for each combination, the information flow requirements will be investigated. Furthermore, advantages and disadvantages for fault handling will be discussed. In the following two sections the different architectures will be described.

### A.4.1 Control System Architecture

A distributed control system consists of a plant to control, and several sensors, actuators, and controllers that are connected by means of a communications network. The controlled plant usually is of a Multi Input Multi Output (MIMO) character. In a distributed control system the sensors and actuators are usually physically constrained to certain spatial positions. This may be denoted the *physical distribution* of the system. On another level we have the *computational decomposition*, which determine how the computational power is distributed. That is equivalent to the computer and communications system distribution, which is treated in the next section. On yet another level we have the *algorithmic decomposition*. This reflects the possible partition of the control problem into smaller subproblems. An example of this is the cascade control structure that is used in many control systems. In the brake system application it can be reasonable to design dedicated brake-force control to each wheel individually, and then use the controlled wheels in the design of a control for the complete vehicle, rather than attacking full vehicle control problem with the uncontrolled wheels directly. The control system architecture in this section refers to the algorithmic architecture rather than the spatial or computational architecture. The algorithmic architectures regarded here are parameterized in the decomposition level. One extreme is the centralized control strategy and the other extreme a strategy consisting of communicating local controllers.

### Centralized Control Strategy

The *centralized* control strategy, see Figure A.1, may be regarded as a monolithic algorithm that collects data from all sensors, and computes outputs to all actuators. In control literature this is the common way to treat the control of MIMO systems. The sensors and actuators are typically read and written frequently. An advantage of this approach is that the control algorithm has all information of the system available when taking decisions on each actuator input. The availability of all sensor data enables the use of state observers. A possible drawback with a large monolithic algorithm is the difficulty to adapt it to changes in the controlled plant. Changes in a small part of the plant may require redesign of the full controller.

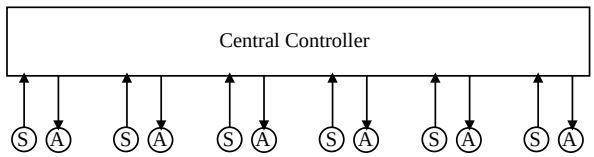


Figure A.1: A centralized control architecture.

### Mixed Control Strategy

Between the extremes there are *mixed* strategies, see Figure A.2, consisting of a central controller that may provide control of top level functionality, and autonomous subcontrollers that control certain aspects of the controlled plant. The central controller may provide the subcontrollers with setpoints and mode switching commands etc. The subcontrollers are responsible for reading sensor data and computing acuator inputs, although the central controller also may handle some of this. Typically the amount of communication required between the central controller and the subcontrollers is lower than for the sensor readings and acuator inputs. To use this approach it is necessary that it is possible to partition the controlled plant into smaller subprocesses with distinct sets of inputs and outputs, which can be controlled more or less independently. Since the resulting controller is of cascade type it is reasonable to design the local control loops to be faster than the central loops. This simplifies the design of the central control loops. A possible advantage of this is simpler adaptation to changes in the controlled plant. It may be so that the necessary changes in the controller are isolated to one subcontroller.

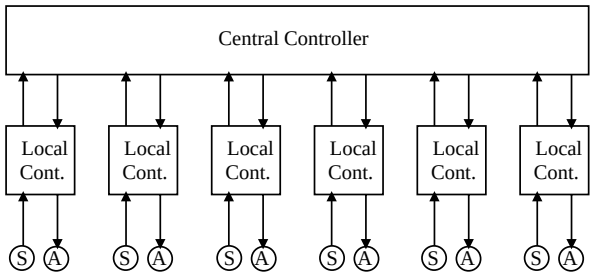


Figure A.2: A mixed control architecture.

### Local Control Strategy

The *local* control strategy, see Figure A.3, consists of a non-hierarchical set of local controllers that communicates. The communication necessary between the controllers are typically low. Here it is necessary that the controlled plant can be partitioned into smaller subprocesses that can be controlled almost independently.

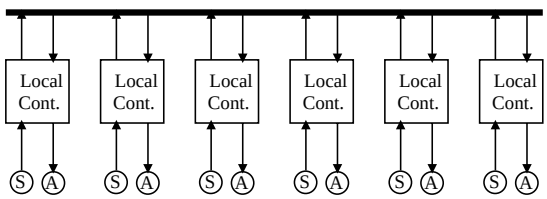


Figure A.3: An local control architecture.

## A.4.2 Computer Architecture

In automotive system such as cars and trucks the amount of cabling is a problem. In order to reduce cabling a multiplexed system is very useful. However, multiplexing does not determine whether the data processing is distributed as well. The data processing may still be centralized, or it may be fully-distributed. Therefore the impact of distributing the computational resources of the computer architecture will be discussed. We concentrate on three levels of distribution, a centralized system, a partially-distributed and a fully-distributed. These three levels will be described below. There exist of course other solutions, however, they will most likely be combinations of those described. Therefore, the discussions

will be concentrated on these three levels and their qualities. Most likely, this will indicate the result we may achieve when using combinations of these solutions.

### Centralized System

In the centralized system only one computer node is used. Sensors and actuators are connected via a buss to the central node. We assume here that a central node with a separate cable to each sensor and actuator will be too costly, both in cabling and space. Therefore, we assume that all sensors and actuators are connected to the central node via a buss, se figure A.4. The sensors and actuators will send and receive data from the bus, respectively. All functionality is located in one computer node, which may be beneficial for maintenance. Furthermore, without redundancy, there are no consistency problems with such a solution, since the central node makes all calculations.

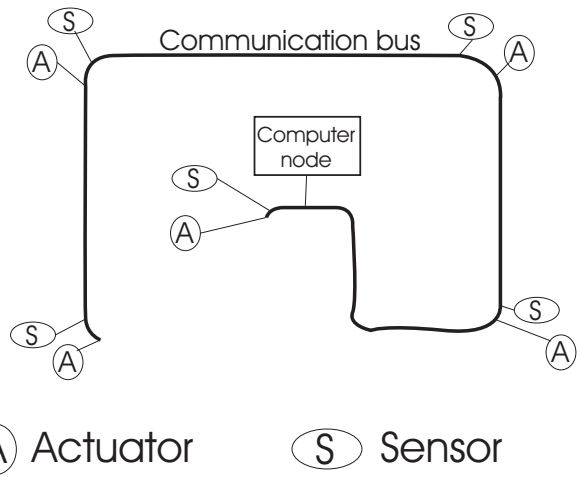


Figure A.4: A centralized computer architecture.

There are a number of specific qualities that can be deduced from the fact that the system is a centralized.

- A centralized system is easier to manage, for example, there will be less problem with consistency with only one central node.

- Less hardware is required, although the system might need a fast processor since all processing is done on one single node.

- Since there is only one master node that will control the communication on the bus. It will be easier to verify that sufficient communication bandwidth exist.

- Maintenance is easier since there is only one node that is placed on a single location.

Despite many good properties, a centralized system may produce a high bus communication load, since all control loops are closed over the communication bus. If the fast inner loops are possible to distribute, i.e., calculations are done at a node with actuators and sensors connected locally, the communication bandwidth can be reduced, see e.g., [TL94]

### Partially-Distributed System

The next level of distribution, which we have distinguished, is when sensors and actuators become more "intelligent," such that some computations are possible at the sensors and actuators. However, the control and computational power will still be handled at a main computer node. Such a system can be viewed in figure A.5, where the sensors and actuators are connected to a simple computer node. The sensor and actuator computer can make simple calculations on sensor values before they are transmitted throughout the system. They may also have the possibility to handle local (inner) control loops. This solution will relieve the main computer node from some of its tasks.

Some of the benefits of making the system partially-distributed is described below.

- When developing the system, it is possible to decompose the system into smaller function units which may be developed in parallel.

- The system will be easier to test since every unit can be tested separately and only an assembly test is required with all units together.

- Smaller units will reduce software complexity. And less complex hardware may be used.

The support for fault tolerance may be more complex, for example, since nodes must know which other nodes are functioning. However, similar requirement may exist on the centralized system, where knowledge are required on which sensors and actuator are alive.

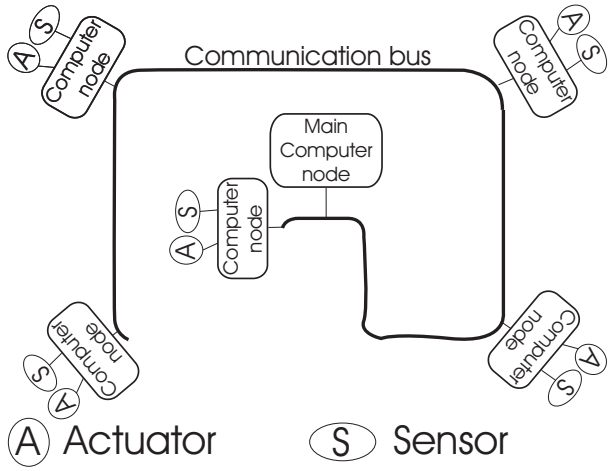


Figure A.5: A partially-distributed computer architecture.

### Fully-Distributed System

The third level of distribution we have identified is the fully distributed. In such a system no computer node has any unique position like the central node in the partially-distributed and central system. All calculations and logic are distributed on computer nodes, see figure A.6, with local sensors and actuators. They have to co-operate with each other in such away that necessary data is provided to all nodes. Thus, no node will have as high load as when using a single centralized node. Although the complexity might increase there are more possibilities to include new functionality.

The benefits of making the system fully-distributed is described below and are similar to the partially-distributed system but more refined. In this case the "modularity" will increase even more since the functional decomposition it stricter.

- When developing the system it is natural to decompose the system into smaller function units which may be developed in parallel.
- These units can then be tested separately and an assembly test is required to ensure correct interfacing among units.
- Smaller units will reduce software complexity. And less complex hardware may be used.
- A node failure will most likely only disable part of the system compared to the other systems where the central node makes them more

sensitive.

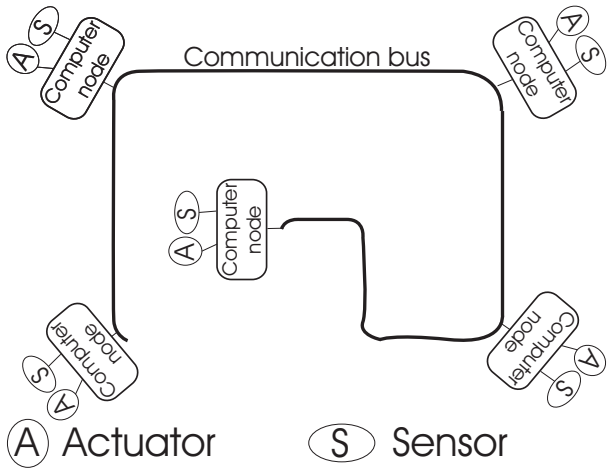


Figure A.6: A full-distributed computer architecture.

## A.5 Dependability of Different Computer Configurations

In this section we will investigate the reliability of the different computer architecture hardware-configurations. This will be done by comparing the described computer architectures with each other.

Three different configurations will be discussed for each computer architecture type. Those are two simplex configurations and one duplex. The first simplex configuration will include all functionality and give the reliability for the total yaw-control system. The other simplex configuration will only include the critical functionality, i.e., functionality necessary for basic braking. The third configuration we study is a duplex system for total functionality.

When making reliability calculations it is difficult to find reliable estimations of the failure rates. One of the most common source is the United States Department of Defence (USDOD) "Military handbook reliability prediction of electronic equipment, MIL-HDBK-217F" [MIL92]. Although it is common to use failure rates from this handbook, this method is criticized for not being trustworthy. The absolute figures generated from this handbook can therefore be questioned. However, the results achieved will only be used for comparing the different solutions, for which purpose the figures from the MIL-HDBK-217 are appropriate [SS92].

In the normal-life portion of the electronic components that we consider, the failure rate $\lambda$ is constant. The reliability expression will then be of the form $R(t) = e^{-\lambda t}$, i.e., exponential. In the reliability calculations we use combinatorial methods such as series/parallel system, markov models are also used and further information on reliability calculations using these methods can be found in for example [Pra95] or [Joh89]. Specific assumptions of each configuration will be described under corresponding section. The failure rates used in this report and assumptions needed are described in Appendix B.2. The reliability analysis is based on our case study system, i.e., yaw-control, ABS and EBS. A similar system study has been conducted on an electrical flight control system in [ATJ99].

### A.5.1 Assumptions

The first configuration is a simplex system where we calculate the reliability for the yaw-control functionality, i.e., all considered functionality. All components must work in the system in order to provide the total functionality of the yaw-control system.

The yaw-control system is safety-critical. However, there exists a fail-

safe state, which is when the yaw-control functionality is shut down and only normal brakes are accessible. Thus when a fault is detected that affects the yaw-control functionality, the yaw-control is shut down and the truck runs in a degraded mode. This degraded mode only has the basic brake functionality.

In the continuation of this document we will refer the full yaw control functionality as the "total functionality" or "yaw-control functionality" in contrast to a degraded mode where parts of the yaw-control are inaccessible.

This leads us to the next configuration, where the reliability of critical functionality is calculated. The critical functionality is considered a basic brake function, where at least one wheel on each side of the truck is working. Thus, the minimum of components needed, for the critical functionality, is the pedal sensor, some basic computational power, and, at least one brake pressure actuator at each side of the truck. Thus when discussing reliability of this critical configuration where at least the basic brakes is functioning, we will use the term "critical functionality".

For the duplex system, we assume that only the critical functionality will be duplicated. Therefore, we only consider and calculate the probability of loss of critical functionality. Inherent redundancy will be used if possible for the reliability analysis of critical functionality, for example, when considering that more than one wheel per side can be used for braking.

In dependable systems, adding redundant software and hardware often increases the complexity of the system. This redundancy must be handled and sometimes reconfigured in case of failures. There exists a number of techniques for redundancy management. Although only using a duplex system we can still use different solutions for the redundancy handling, i.e., active or passive redundancy as discussed in [SCG00]. The complexity of a duplex system will increase independently of the redundancy management technique. For example, with two redundant nodes both the active and passive redundancy requires fail silent nodes. Therefore, a lot of effort must be spent to ensure the fail silent property with sufficient probability. Furthermore, a membership agreement protocol is required with a passive redundancy strategy, and with active redundancy, the redundant node must produce the same output. To reduce the complexity of our investigation, the redundancy methods requiring the most resources are used for the duplex system. The active redundancy strategy generally requires more resources than a passive system, since then redundant components are executing simultaneously. Furthermore, computation should be deterministic to ensure identical output from replicas.

A special comment on the bus failure rate, $\lambda_{Bcon}$, is necessary. The

bus failure rate has been divided in two parts. The first is a failure mode with failure rate $\lambda_1$, which is considered by far the most probable. This failure occurs when a node cannot communicate any more with other components connected to the bus. The second, is when a buss connector prohibits any component from communication on the bus. This scenario is considered a far less probable case, with failure rate $\lambda_{Bof}$, but a more serious one. Further bus failure modes, like partitioning of the bus, have not been considered specifically but are considered part of $\lambda_{Bof}$. This is a bit conservative since the system may still work.

The calculations in Appendix B.2 use data from the MIL-HANDBK [MIL92]. Since, this handbook does not cover sensors and actuators. Therefore, rates for sensors and actuators has been estimated to be between $10^{-6} - 10^{-4}$ failures per hour. In this section a failure rate of $15 \times 10^{-6}$ will be used in the discussions if nothing else is stated, in order to give directly comparable results. For further details see Appendix B.3.

The theories for the calculations here and in Appendix B.2 can be found in [Joh89].

### A.5.2 Reliability of Total Functionality - Simplex

The reliability of the total functionality gives an indication of the systems availability, i.e., the availability of the yaw-control, ABS, and brake system. With this case we compare the different architectures considering the reliability of the total functionality. We assume that if any component fails, it will affect the system such that the yaw-control will cease to work properly.

For the central system we use the configuration shown in Figure A.7. The picture shows the nodes, sensors and actuators connected to the buss. The basic components, of node and sensors/actuators, that are considered in the reliability analysis are also shown in the figure, see also Appendix B.2. The system consists of a central computer node where all the calculations are done. The sensors in the system are the same in all configurations and consist of the pedal sensor, a steering wheel sensor, a yaw rate sensor, a lateral acceleration sensor, and one wheel velocity sensor per wheel (6 wheels). Finally, we have brake pressure actuators at each wheel, which also are the same for all configurations. In the centralized system, sensors and actuators are more complex since they need functionality and hardware for the bus connection, which affects the failure rate.

For the partially-distributed and fully-distributed system the configurations are shown in Figure A.8. These systems have the same set of sensors and actuators. The main difference is the central node which is

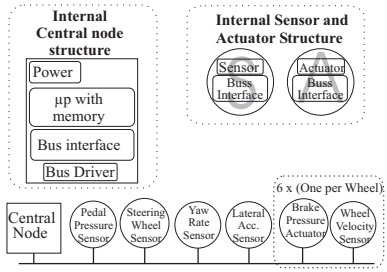


Figure A.7: A simplex centralized computer architecture.

only used in the partially-distributed system. Another difference is the complexity of the distributed nodes, which are more complex in the fully-distributed system. These differences also affect the failure rate of the system.

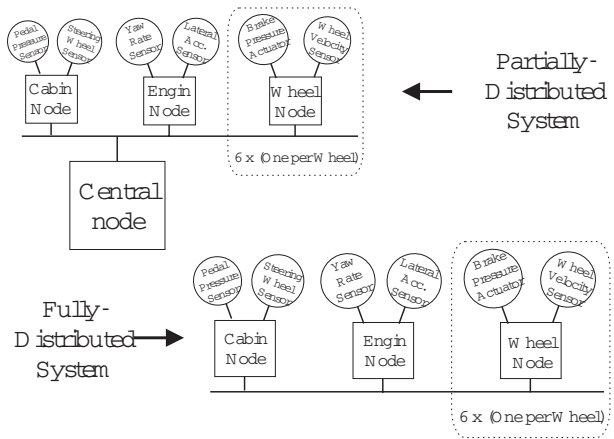


Figure A.8: partially-distributed and fully-distributed architectures.

In Appendix B.3 the probability of loss of yaw-control functionality, i.e., the total functionality, has been calculated for one hour. This probability is presented in Table A.1 for the different configurations. It should be noted, that the difference between the architectures are mainly due to: the hardware complexity of the different computer nodes, complexity of sensors and actuators, how sensors and actuators are connected, and, the number of connections to the bus. Due to the difficulties of finding reliable values for failure rates of sensors and actuators they have been varied between $10^{-6}$ and $40 \times 10^{-6}$. However, for the purpose of comparison, the values presented in Table A.1 have been calculated with the

failure rate $\lambda_s = \lambda_a = 15 \times 10 - 6$.

| Cent. Sys. | Part. Dist. Sys. | Dist. Sys. |
|---|---|---|
| $3.3 \times 10^{-4}$ | $3.5 \times 10^{-4}$ | $3.5 \times 10^{-4}$ |

Table A.1: Probability of loss of the yaw-control functionality, in one hour

These figures are in the same order of magnitude. Any specific preferable solution cannot be identified. In Figure A.9, the same probabilities are shown when the failure rate of the sensors and actuators are varied between $10^{-6}$ and $40 \times 10^{-6}$. In this figure, we see how close these configurations are. The continuous line, of the centralized system, is the line with lowest probability of failure, followed by the partially-distributed and fully-distributed systems which are superposed.

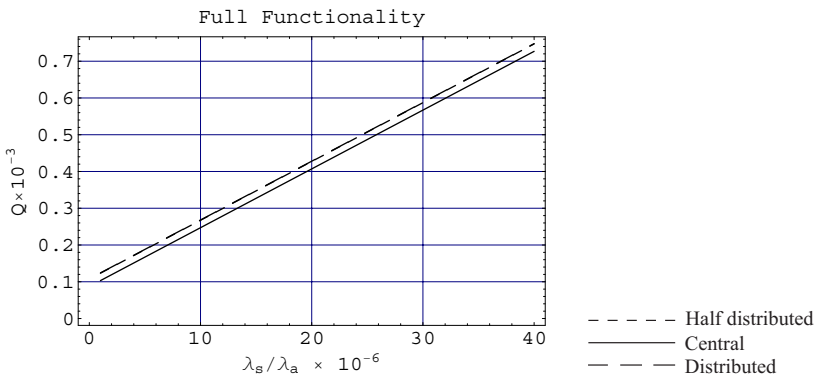


Figure A.9: Probability of losing the yaw-control functionality for the different configurations.

To analyze the effects of the complexity of nodes, we have compared the different configurations while changing the ratio between the complexity of the processors in the nodes. In Appendix B.2, we have used the MIL-HANDBK [MIL92] and the estimated workload on the nodes to get the failure rate of processors. The workload will set the required complexity of a node, i.e., the performance, to handle the required workload. Uncertainty of the complexity relation between nodes originates from the processing requirements of the different node types, i.e., central node, partially-distributed nodes, and distributed nodes. The complexity directly influence the failure rate of the processor. To investigate the effect of this complexity ratio we have varied the distributed nodes failure rate while the failure rate of the central node has been fixed. The

complexity of fully-distributed and partially-distributed node processors have been changed between 20-100% of the central node. In Figure A.10 we see how the reliability of the fully-distributed and partially-distributed configuration decrease when the complexity increases. In our calculations in Appendix B.3 we have assumed that the failure rate ratio between the processors in the partially-distributed nodes and the central nodes are 30%. Between the processors in the distributed nodes and the central the ratio is 50%. Figure A.10 shows the reliability of each processor, where the failure rates of the sensors and actuators are varied. The figure shows that if the complexity of the partially-distributed and fully-distributed nodes is small enough, the system reliability will be close to that of the centralized system. If the complexity of the fully-distributed nodes are very low they become similar to simple "intelligent" sensor and actuator nodes used in the central configuration. As the complexity of the fully-distributed nodes increase, the reliability is decreasing compared to the central system.

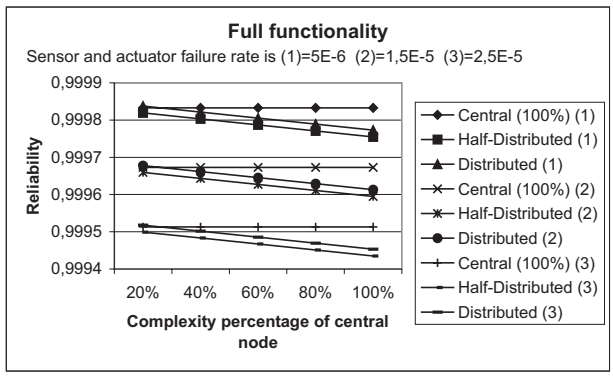


Figure A.10: The reliability of different configurations when varying the complexity relation of the partially-distributed and fully-distributed nodes to the central node.

### A.5.3 Reliability of Critical Functionality - Simplex

In this section the reliability of the critical system parts will be discussed. The basic brake functionality is here considered the critical part of the system. It is assumed that the absolute minimum allowed braking functionality is brake force on two wheels, i.e., braking on one wheel on each side. This implies that the required functionality is "one brake pressure actuator on each side of the vehicle". Since there are three wheels on each side, there is an inherent redundancy in the system. The critical

components needed for basic braking are the pedal sensor, some basic computational power, and one brake pressure actuator on each side.

For the central system, the central node must provide the computational power. However, in the partially-distributed system, the computational power in the distributed nodes is considered sufficient for the basic processing needed. The minimum system parts needed for the different systems configurations are shown in Figure A.11.

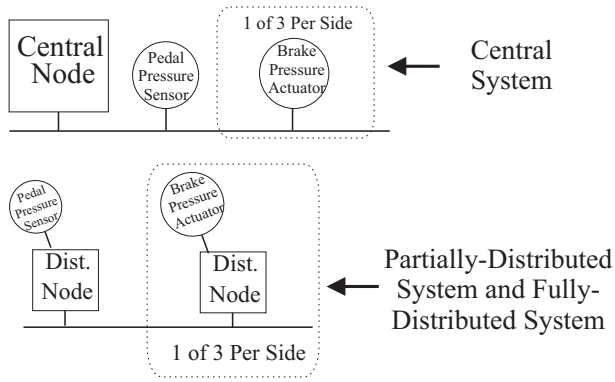


Figure A.11: Minimum system parts needed to achieve critical functionality.

The probability of loss of the critical functionality will naturally be less than in the case with total functionality. In Table A.2, values for the total yaw control functionality and the critical functionality are shown for comparison.

| | Cent. Sys. | Part Dist. Sys. | Dist. Sys. |
|---|---|---|---|
| Full Func. | $3.3 \times 10^{-4}$ | $3.5 \times 10^{-4}$ | $3.5 \times 10^{-4}$ |
| Critical Func. | $4.3 \times 10^{-5}$ | $2.7 \times 10^{-5}$ | $3.0 \times 10^{-5}$ |

Table A.2: Probability of loss of critical functionality in one hour for the different configurations.

The probabilities of loss of critical functionality is about a factor of ten better than for the total functionality. In Figure A.12 the probability of loss of critical functionality is shown when the failure rate for sensors and actuators is varied.

The central system has a noticeable higher probability of critical failures, the continuous line in the figure. This because, the central node must be present and the central node has a higher failure rate than the

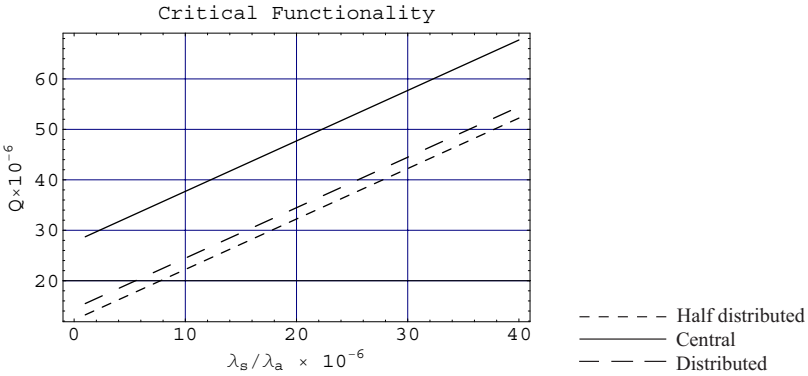


Figure A.12: Probability of loss of critical functionality with the different configurations as a function of sensor/actuator failure rate.

nodes in the partially-distributed and fully-distributed system. Furthermore, the central node has more bus connections, which increases the failure rates of the bus in the central system.

The partially-distributed and fully-distributed systems have similar probability for critical failure. They have very similar configurations in this case. The small difference is due to the difference in complexity of the nodes. Since the fully-distributed systems nodes are more complex, they also have the higher probability of failure.

In the complexity comparison we use a fixed failure rate for the central node while varying the failure rate for the fully-distributed and partially-distributed architectures. The processor complexity of the fully-distributed and partially-distributed nodes have been varied between 20-100% of the central node processor. In Figure A.13 we see the value of using low complexity nodes for critical functionality. Only the central and partially-distributed system have been included in this figure since the partially-distributed and fully-distributed system follow the same trend and would be very close.

### A.5.4 Reliability of Duplex System Configurations

The duplex configurations have been chosen to increase the reliability. However, to keep the cost as low as possible only the parts needed for the critical functionality will be duplicated, i.e., the parts needed for a minimum of brake functionality. Furthermore, parts that can benefit from the inherent redundancy in the system have not been duplicated. The brake pressure actuators will therefore not be duplicated; neither will the nodes that the actuators are attached to. However, the brake pedal

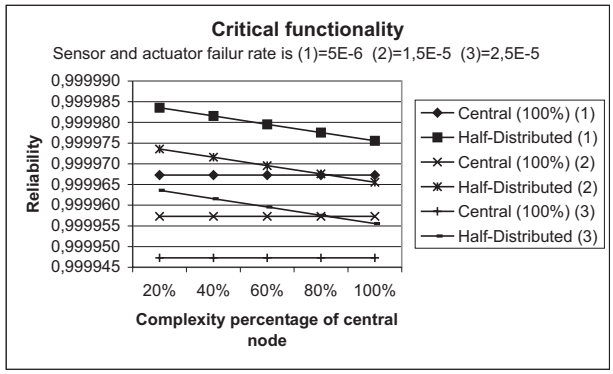


Figure A.13: The reliability of critical functionality for the different configurations when varying the complexity relation of the partially-distributed to the central node.

sensor will need special treatment, since it has been considered to be an especially critical part. To ensure brake-pedal input from the driver, redundancy will be used with the brake-pedal sensor. The brake-pedal sensor is not assumed to be fail silent and is therefore triplicated, thus one faulty sensor is allowed.

For the central system, the central node and the bus will be duplicated. Three redundant pedal sensors will be connected to the bus. The partially-distributed and fully-distributed system will use three pedal sensors connected to a duplicated node. The difference is again the complexity of the nodes. In Figure A.14 the duplex configurations are shown for the three different architectures.

The duplicated nodes may fail in two ways, (1) a node fails but do not disturb the operation of fault free parts of the system. (2) The node fails in such a way that it affects other parts of the system and the required functionality can not be upheld. Scenario (2) is naturally very important to avoid, and considerable efforts should be spent ensuring that faults do not spread through out the system. This is done by introducing containment regions, which should be designed such that no faults can escape such a region. One related property in this situation is fail silence, see [SCG00], where a component either produces the correct result or no result at all. To achieve fault containment, e.g., fail-silence, the system must be able to detect and handle faults. For a fail-silent system this means to stop producing and sending results. The coverage of the error detection mechanisms is important and will affect the system reliability. However, to be able to conduct the reliability calculations the coverage must be estimated. In Appendix B.3, the coverage has been

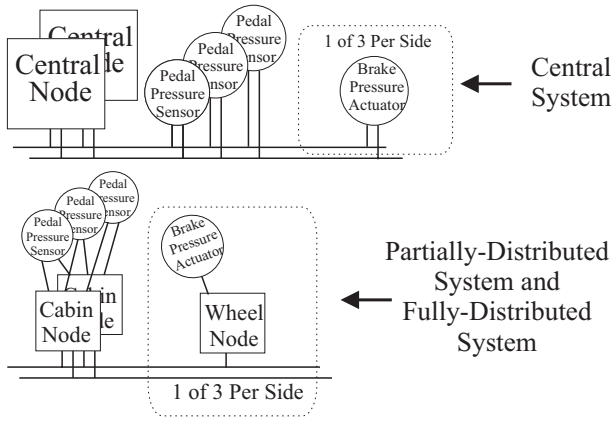


Figure A.14: System parts for duplex configurations where critical parts are duplicated. The wheel nodes and brake pressure actuator use the inherent redundancy; three wheels on each side of the truck.

between 0.99 and 0.999. This is a high coverage but indications that a very high coverage may be attained is given in, for example, [Fol99]. Furthermore, we assume that all efforts are spent ensuring this coverage, both in the system and application level of the design. In the table below the importance of the coverage will be shown.

When adding redundancy to these system configurations the reliability increases significantly. In Table A.3 the probability of loss of the critical functionality for the duplex system configurations are shown. Values for total functionality and critical functionality in the simplex configurations are shown for comparison. The failure rate for sensors and actuators is as earlier $\lambda_s = \lambda_a = 15 \times 10^{-6}$.

| | Cent. Sys. | Part. Dist. Sys. | Dist. Sys. |
|---|---|---|---|
| Full Func. (simplex) | $3.3 \times 10^{-4}$ | $3.5 \times 10^{-4}$ | $3.5 \times 10^{-4}$ |
| Critical Func. (simplex) | $4.3 \times 10^{-5}$ | $2.7 \times 10^{-5}$ | $3.0 \times 10^{-5}$ |
| Duplex (C=0.99) | $3.7 \times 10^{-7}$ | $2.2 \times 10^{-7}$ | $2.7 \times 10^{-7}$ |
| Duplex (C=0.995) | $1.8 \times 10^{-7}$ | $1.1 \times 10^{-7}$ | $1.4 \times 10^{-7}$ |
| Duplex (C=0.999) | $3.8 \times 10^{-8}$ | $2.2 \times 10^{-8}$ | $2.7 \times 10^{-8}$ |

Table A.3: Probability of loss of critical functionality in duplex configurations. The probabilities are calculated using coverage, C, of 0.99, 0.95, and 0.999. Probabilities for full functionality and critical functionality are included for comparison.

The differences between the three architectures are basically due to the same reason as for the simplex case with only critical functionality, the differences of node complexity and the bus. However, especially interesting is the importance of achieving a high coverage during system design.

The probability of loss of critical functionality is about a factor of 100 to 1000 better with the duplex system compared to the simplex. In Figure A.15 the probability of loss of critical functionality is shown, when the failure rate for sensors and actuators is varied.

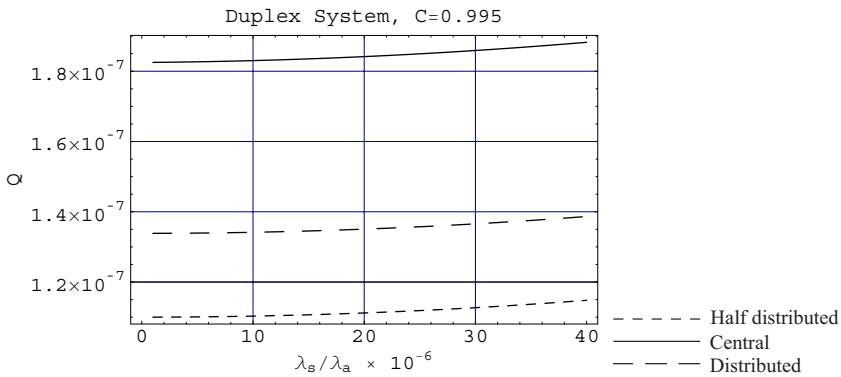


Figure A.15: Probability of loss of functionality with the different duplex configurations.

The results from the sensitivity analysis is shown in Figure A.16. In this figure we see a considerable change in reliability when the complexity ratio of the nodes changes. We see here that the effect of complexity is even larger when considering a duplex system, i.e., we gain even more in reliability if the complexity is kept low for nodes in a duplex system. The fully-distributed system is not included since the difference between the fully-distributed and partially-distributed system is very small in this comparison.

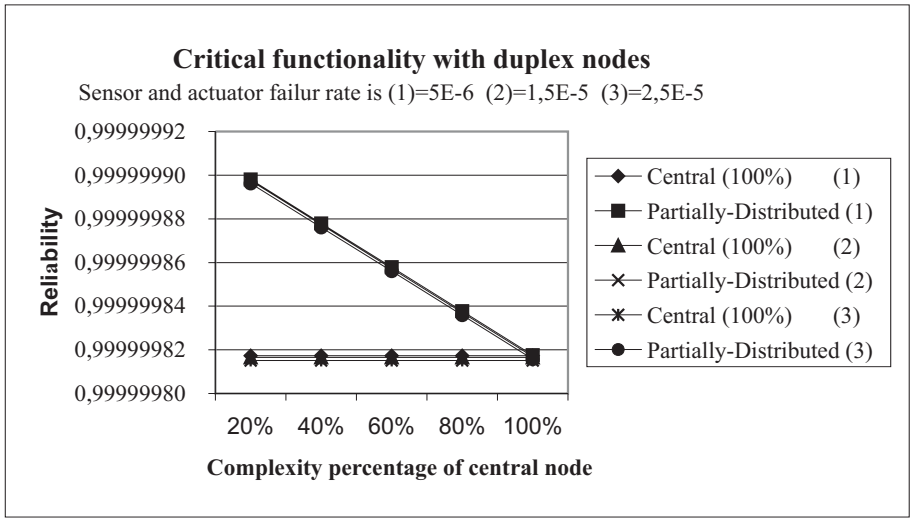


Figure A.16: The reliability of critical functionality with duplex nodes, i.e., with redundancy. The partially-distributed configuration are compared with the centralized by varying the complexity of the nodes in the partially-distributed system between 20-100% of the complexity of the node in the centralized system.

## A.6 Architecture Investigation

In the following sections we will compare the different combinations of computer and control architectures. We will discuss the effect on the worst case bandwidth requirements as well as the dependability, when combining the different control architectures with the different computer architectures. Worst case communication bandwidth is required when a system runs with full-functionality, thus we will *not* consider the critical functionality when studying bandwidth requirements. There were three computer architectures described in section A.4, with different distribution levels, and similarly a description of three control architectures. The different combinations can be viewed in a 3x3 matrix, as shown in Table A.4. The table consists of references to the section where corresponding combination are treated. All architectural combinations are not reasonable and will thus only be treated briefly stating why they are not appropriate.

| | Centralalized | partially-distributed | Distributed |
|---|---|---|---|
| **Cent. control** | A.6.2 | A.6.3 | A.6.4 |
| **Mixed control** | A.6.5 | A.6.6 | A.6.7 |
| **Local control** | A.6.8 | A.6.9 | A.6.10 |

Table A.4: Control and architecture matrix with section references.

### A.6.1 Assumptions

To make a reasonable comparison between these architectures we need a realistic workload and environment. In the companion report "Investigation and Requirements of a Computer Control System in a Heavy-Duty Truck" [SCG00] we have investigated an existing truck solution. In this report we have presented an appropriate communication workload based on the existing system. The computer control systems discussed in this report are intended for future computer controlled trucks. Therefore, we have included functions that we predicted likely to be used in futures systems. The characteristics of these applications have been estimated based on the investigation in [SCG00], such as, bandwidth, sensors, sensor location, etc. Thus, when estimating the bandwidth in the sections below, we have tried to predict data that will be sent on the bus in future systems. In order to get a realistic bus load more components has been considered compared to what has been discussed in the dependability analysis of section A.5. Specifically, additional sensors and actuators are used.

In Table A.5 and A.6, sensors which we have assumed to be connected to the distributed computer system in the heavy-duty truck are listed, these are taken from Appendix A and B in [SCG00] where corresponding messages are listed. The first list includes all single sensors and the second list includes sensors which are located at each wheel.

Some flags, like ABS active flag in the Appendix A & B in [SCG00], are assumed to be part of messages shown in Table A.6. Actuators present in the system are listed in Table A.7.

The retarder works on the gear box by hydraulic or electro magnetic forces. The electro magnetic retarder can also be placed between the gearbox and the rear axle. The engine brake basically works by restraining the exhaust in the cylinders.

In this communication bandwidth study we have assumed that the communication bandwidth is a limited resource. This will probably be true in the foreseeable future, at least in this type of applications. However, other aspects exists and should not be overlooked, for example:

- Sending all control data on the bus should not only be seen as an

| Sensor | Type | Rep/Min [ms] | Size [bits] |
|---|---|---|---|
| Sensor for lateral acceleration | P | 100 | 16 |
| Steering wheel sensor | P | 50 | 16 |
| Yaw rate sensor | P | 100 | 16 |
| Articulation angle | P | 100 | 16 |
| Engine Torque sensor | P | 10 | 16 |
| Engine rpm sensor | P | 10 | 16 |
| Vehicle speed, (Virtual sensor) | P | 20 | 16 |
| Accelerator pedal sensor | P | 50 | 16 |
| Brake pedal sensor | P | 50 | 16 |
| Gear selection sensor | S | 100 | 8 |
| Trailer brake pressure sensor* | P | 20 | 16 |

Table A.5: Sensors connected to a vehicle dynamic computer system in a heavy-duty truck. P = produce Periodic messages, next column the Repetition, and S = produce Sporadic messages, next column is Min interarrival time. The Articulation angle measures the angle between truck and trailer, in the horizontal plane

| Sensor | Type | Rep/Min [ms] | Size [bits] |
|---|---|---|---|
| Wheel speed sensor | P | 20 | 16 |

Table A.6: Sensors connected to a vehicle dynamics computer system in a heavy-duty truck and which exist for each road wheel. P = produce Periodic messages, next column the Repetition.

| Actuators | Type | Rep/Min [ms] | Size [bits] |
|---|---|---|---|
| Engine torque actuator | p | 10 | 16 |
| Brake pressure actuator, one per wheel | p | 10 | 16 |
| Retarder actuator (Hydraulic) | p | 10 | 16 |
| Retarder actuator (Electro magnetic) | p | 10 | 16 |
| Engine brake actuator. | p | 10 | 16 |
| Trailer brake pressure actuator | p | 10 | 16 |
| Rear wheel steering actuator | p | 50 | 16 |

Table A.7: Actuators used in the vehicle dynamics computer system in a heavy-duty truck. P = consume Periodic messages, next column the Repetition.

disadvantage. It gives one significant advantage since it simplifies monitoring of a system.

- It is also more future proof, in the sense that all data already exists on the bus for new functionality to use.

In the following sections we will look at the different combinations and identify which data is sent.

Two communication bandwidth calculations will be done for each of the relevant combinations of control and computer architectures. For each of them we will study the communication in a simplex system and a duplex system. In the simplex case no redundancy is assumed and for the duplex case critical parts are redundant. This simplex case corresponds to the simplex configuration with total functionality which was considered in our dependability calculations earlier. The second case with critical functionality will be duplicated, corresponds to the duplex configurations earlier.

An additional node has been added, a sensor node, which was not included in the reliability study. This has been done since the uncertainty to which node these sensors would be connected to in a real system. This node has been added in all configurations, except the centralized, such that all configurations have the same prerequisites.

### A.6.2 Central Strategy

With a combination of the centralized control strategy and the centralized computer architecture we have a natural centralized system. This has traditionally been a common solution with one node handling all functionality. All sensors and actuators are connected to this node and often with separate cables. However, one useful way of reducing cabling is to use multiplexing on a single communication bus. The bus reduces cabling at the cost of some extra complexity at each sensor and actuator. All control execution and system monitoring are executed at the single central node. With this system, bandwidth requirements will be studied for our case study application.

We also note that for this configuration all sensor and actuator data are sent on the communication bus and are available to, for example, diagnostic units listening on the bus.

#### Simplex System

Using this strategy *all* sensor and actuator data are sent on the bus. Table A.17 describes the assumed bandwidth requirement excluding all overhead.

| **Sensors** | **bits** | **#** | **Tot.** | **Hz** | **Bits/sec** |
|---|---|---|---|---|---|
| Sensor for lateral acceleration | 16 | 1 | 16 | 10 | 160 |
| Steering wheel sensor | 16 | 1 | 16 | 20 | 320 |
| Yaw rate sensors | 16 | 1 | 16 | 10 | 160 |
| Articulation angle | 16 | 1 | 16 | 10 | 160 |
| Engine Torque sensor | 16 | 1 | 16 | 100 | 1600 |
| Engine rpm sensor | 16 | 1 | 16 | 100 | 1600 |
| Vehicle speed, (Virtual sensor) | 16 | 1 | 16 | 50 | 800 |
| Accelerator pedal sensor | 16 | 1 | 16 | 20 | 320 |
| Brake pedal sensor | 16 | 1 | 16 | 100 | 1600 |
| Gear selection sensor | 8 | 1 | 8 | 10 | 80 |
| Trailer brake pressure sensor* | 16 | 1 | 16 | 50 | 800 |
| Wheel speed sensors X 6 | 16 | 6 | 96 | 50 | 4800 |
| **Total number of bits per second** | | | | | *12400* |
| **Actuators** | **bits** | **#** | **Tot.** | **Hz** | **Bits/sec** |
| Engine torque actuator | 16 | 1 | 16 | 100 | 1600 |
| Brake pressure actuator, one per wheel x 6 | 16 | 1 | 16 | 100 | 1600 |
| Retarder actuator (Hydraulic) | 16 | 1 | 16 | 100 | 1600 |
| Retarder actuator (Electro magnetic) | 16 | 1 | 16 | 100 | 1600 |
| Engine brake actuator | 16 | 1 | 16 | 100 | 1600 |
| Trailer brake pressure actuator | 16 | 1 | 16 | 100 | 1600 |
| Rear Wheel steering actuator x 4 (four RW) | 16 | 4 | 64 | 20 | 1280 |
| **Total number of bits per second** | | | | | *10880* |
| **Total required bandwidth** | | | | | **23280** |
| **Max frequency** | | | | **100** | |

Figure A.17: Bandwidth requirement for a central strategy

In Table A.17 the "bits" column indicates the number of message bits. The "#" column, shows the number of messages sent for each specific sensor or actuator. All sensors and actuators values are transmitted periodically, except the gear selection sensor, which is aperiodic. In the case of the gear selection sensor, we find in the "Hz" column the minimum inter arrival time allowed for the gear selection value.

The total bandwidth requirement of this approach is 23,3 kbit/s. However, this is pure data and all protocol overhead will normally increase this figure significantly. This is the case for all the configurations. Furthermore, only data used in the VDC-system is included in the table.

### Duplex System

For a duplex system, data transfer will increase for replica management reasons. Here we consider the extra data necessary for keeping duplex nodes replica deterministic. As shown in Figure A.14 the central node is duplicated and the pedal pressure sensor is triplicated. To ensure replica deterministic behavior of the central node, the replicas will exchange sensor values. This means that, first all sensor data will be sent; thereafter

each replica will send their view of the sensor values. Thus, sensor values will be sent three times, and the bandwidth will increase accordingly. The additional, i.e., extra, sends will ensure that the replicas have the same set of data and that they agree on the same pedal sensor value.

The design decision, to exchange all data between both the replica nodes, may seem as a drastic measure when only the pedal sensor has previously been considered critical. However, to ensure that the replicated nodes produce the same result, it is very important that they work on the same input. If not all sensor data was exchanged, and one node looses some data, this node would have to be silent. This would be necessary to ensure the fail-silent property.

| **Sensors** | **bits** | **#** | **Tot.** | **Hz** | **Bits/sec** |
|---|---|---|---|---|---|
| Sensor for lateral acceleration | 16 | 1 | 16 | 10 | 160 |
| Steering wheel sensor | 16 | 1 | 16 | 20 | 320 |
| Yaw rate sensors | 16 | 1 | 16 | 10 | 160 |
| Articulation angle | 16 | 1 | 16 | 10 | 160 |
| Engine Torque sensor | 16 | 1 | 16 | 100 | 1600 |
| Engine rpm sensor | 16 | 1 | 16 | 100 | 1600 |
| Vehicle speed, (Virtual sensor) | 16 | 1 | 16 | 50 | 800 |
| Accelerator pedal sensor | 16 | 1 | 16 | 20 | 320 |
| **Brake pedal sensor** | 16 | 3 | 48 | 100 | 4800 |
| Gear selection sensor | 8 | 1 | 8 | 10 | 80 |
| Trailer brake pressure sensor* | 16 | 1 | 16 | 50 | 800 |
| Wheel speed sensors X 6 | 16 | 6 | 96 | 50 | 4800 |
| **Total number of sensor data** | | | | | *15600* |
| **Total number of sensor data x 3** | | | | | *46800* |
| **Actuators** | **bits** | **#** | **Tot.** | **Hz** | **Bits/sec** |
| Engine torque actuator | 16 | 1 | 16 | 100 | 1600 |
| Brake pressure actuator, one per wheel x 6 | 16 | 1 | 16 | 100 | 1600 |
| Retarder actuator (Hydraulic) | 16 | 1 | 16 | 100 | 1600 |
| Retarder actuator (Electro magnetic) | 16 | 1 | 16 | 100 | 1600 |
| Engine brake actuator | 16 | 1 | 16 | 100 | 1600 |
| Trailer brake pressure actuator | 16 | 1 | 16 | 100 | 1600 |
| Rear Wheel steering actuator x 4 (four RW) | 16 | 4 | 64 | 20 | 1280 |
| **Total number of acturator data** | | | | | *10880* |
| **Total required bandwidth** | | | | | **57680** |
| **Max frequency** | | | | **100** | |

Figure A.18: Bandwidth requirement for a central strategy with redundancy

In Table A.18 the worst case bandwidth requirements for the central strategy with redundancy is shown, i.e., worst case implies with full functionality running. This combination is resulting in a required bus bandwidth of 57.7 kbit/s. This is more than double the amount than in the simplex case. It indicates the high overhead that replication may lead

to.

### Fault Scenarios

In this section we will have a short discussion about effects of faults in this type of a system. Only the redundant system will be discussed, since, in the simplex case, all functionality will be lost if the central node fails. When subject to a transient fault, less critical parts may of course use restart. However, for this report we have consider the outage time following from a restart as unacceptable for the critical brake system.

If there is a fault in one of the redundant nodes it is very important to prohibit the fault from propagating. It has to be obvious for the actuators, i.e., the receivers, which node sends correct values. For this purpose the fail-silent property is beneficial and has been assumed. In section A.5.4, the importance of high coverage on the fail-silent property has been shown.

In case of transient faults, we are confronted with another challenge, how to handle re-integration of a node that detects an internal fault. If this fault has not caused the node state to change, the node can continue its operation. If the fault has caused different states in two replicas , the re-integrating node has to obtain the correct state information from its replica. Even though the nodes exchange state information during normal operation, e.g., by exchanging sensor values, a central node will need additional state information to get the same state as its replica. This can be a considerable amount of information which must be exchanged. This is a drawback, since the necessary communication bandwidth for the state transfer must be scheduled such that, independent on other communication the transfer will be possible within a limited time. If this is not done re-integration of nodes will not be possible.

In case of a sensor or actuator failure it is very important that the faults are detected. This because a faulty sensor, e.g., the yaw rate sensor, may produce faulty data which potentially puts the vehicle in a dangerous situation. Similarly, if an actuator fails it is very important for the system to detect this. Then the error can be compensated for, especially in the case of a brake pressure actuator. Thus, faults should be detected and appropriate measures should be taken, e.g., reconfiguration or shutting down functionality. This will require some group membership handling or control system diagnosis, in order for the central node to keep track of which sensors and actuators are alive. Unfortunately, this will increase the requirement on sensor and actuator functionality and thus the complexity. This would increase the failure rate of sensors and actuators, which would make them closer in complexity to the distributed nodes in the partially-

distributed system.

### A.6.3 Central Control with Partially-Distributed Computer Architecure

This is not an obvious design solution since the distributed nodes are used for very little. The main difference is that we assume that the duplicated node may handle a situation where the central node fails. The duplicated node is the cabin node, see Figure A.14. Therefore, it is not necessary to make the central node duplicated. If the central node fail, the cabin node has to reconfigure itself to handle the critical functionality, i.e., the basic brake functionality.

In this system we still have the advantage that comes from that all sensor and actuator data are sent on the bus, see section A.6.2.

#### Simplex System

This configuration will still use the central node for all control computations since it uses the central control strategy, see Figure A.1. Therefore, all sensor and actuator data has to be sent on the bus. This makes the amount of bus communication for the simplex case identical to the one in the previous section, see Table A.17.

#### Duplex System

In the duplex case the cabin node is replicated, see Figure A.14. This means that the replicas of the cabin node will agree on which sensor values to use. The replicas will therefore send their sensor values twice. One time to get agreement with its replica and the other time to send the agreed value. Table A.19 shows the bandwidth required for this case. Messages in the table are sorted under the node which sends it.

The agreement messages sent by the cabin node are referred to as Replica Deterministic (RD) messages. The cabin node sends each message once for agreement and once the agreed message. This is done for each replica. Thus in total each message is sent four times, which also is indicated in the table.

As can be seen the bandwidth is considerably less than for the centralized scenario.

#### Fault Scenarios

In this case the central node is not redundant, therefore, functionality running on this node will be lost in case of a permanent central node

| Sensors | bits | # | Tot. | Hz | Bits/sec |
|---|---|---|---|---|---|
| **Central node** | | | | | |
| Brake pressure actuator | 16 | 1 | 16 | 100 | 1600 |
| Engine torque actuator | 16 | 1 | 16 | 100 | 1600 |
| Retarder actuator (Hydraulic) | 16 | 1 | 16 | 100 | 1600 |
| Retarder actuator (Electro magnetic) | 16 | 1 | 16 | 100 | 1600 |
| Engine brake actuator | 16 | 1 | 16 | 100 | 1600 |
| Rear Wheel steering actuator x 4 (four RW) | 16 | 4 | 64 | 20 | 1280 |
| Trailer brake pressure actuator x 1 | 16 | 1 | 16 | 100 | 1600 |
| **Total number bits** | | | | | *10880* |
| **Cabin node x 2 (RD messages)** | | | | | |
| Brake pedal sensor | 16 | 4 | 64 | 100 | 6400 |
| Steering wheel sensor | 16 | 4 | 64 | 20 | 1280 |
| Accelerator pedal sensor | 16 | 4 | 64 | 20 | 1280 |
| Gear selection sensor | 8 | 4 | 32 | 10 | 320 |
| **Total number bits** | | | | | *9280* |
| **Sensor node** | | | | | |
| Sensor for lateral acceleration | 16 | 1 | 16 | 10 | 160 |
| Yaw rate sensors | 16 | 1 | 16 | 10 | 160 |
| Articulation angle | 16 | 1 | 16 | 10 | 160 |
| **Total number bits** | | | | | *480* |
| **Engin node** | | | | | |
| Engine Torque sensor | 16 | 1 | 16 | 100 | 1600 |
| Engine rpm sensor | 16 | 1 | 16 | 100 | 1600 |
| **Total number of sensor data** | | | | | *3200* |
| **Wheel nodes x 6** | | | | | |
| Wheel speed sensors | 16 | 6 | 96 | 50 | 4800 |
| Trailer brake pressure sensor* | 16 | 1 | 16 | 50 | 800 |
| **Total number of sensor data** | | | | | *5600* |
| **Total required bandwidth** | | | | | **29440** |
| **Max frequency** | | | | **100** | |

Figure A.19: Bandwidth requirement for a central control and partially-distributed architecture with redundancy.

failure. However, we assume that the distributed nodes have the possibility to reconfigure, if the central node stops in such a way that the critical functionality will continue working. What we achieve with this is a higher reliability for critical functionality. However, we add software complexity for the reconfiguration. This reconfiguration requires fast detection of central node failure, in order to make the reconfiguration fast. This requires a group membership service for the cabin and the central node. The allowed outage time for the critical functionality, i.e., the time the system can manage without new sensor information, will decide if reconfiguration is possible.

The cabin node will however be vital both for critical and full functionality since it hosts the pedal sensors. For the reasons above we can see that the reliability of the system will increase considerably by making

the cabin node a duplex node, i.e., consisting of two replicas. By doubling the bus, we eliminate a single point of failure in the critical functionality.

For the duplex system we only assume that the cabin node is duplicated. This gives good reliability since we do not have to rely on the complex central node. However, since the central node is not duplicated, the availability of the total functionality is less than the system with centralized control and centralized computer architecture in Section A.6.2.

The necessary state information exchange between replicas after a transient fault, is very small for this configuration. The centralized control ensures that no or very little functionally is run on the replicas.

The importance of distinguishing between correct and incorrect replicas is still high. When using a duplicated cabin node those replicas must be fail-silent in order for the other nodes to choose correct sensor data. Nodes that are not duplicated must fail in a safe manner. For example, a wheel node is not allowed to apply an arbitrary brake force. The recovery after a transient fault will be fast compared to the duplex nodes since no transfer of system states is necessary.

### A.6.4 Central Control with Fully-Distributed Computer Architecture

This combination is not considered here for two reasons. First, as we discussed in the reliability analysis the distributed nodes are not as powerful, thus all control functionality might not be scheduled in the distributed computer nodes. Second, the conceptual difference between this configuration and the local control together with the distributed system is small or none, and bandwidth requirements are the same, see section A.6.10. With the development of microprocessors of today, computational power of processors are unlikely to be a limiting factor.

### A.6.5 Mixed Control with Central Computer

This combination is not considered, since all computation must be in the central node. Therefore, it is no noticeable difference, for our bandwidth investigation, compared with the central control together with the central computer architecture. The difference is how the control architecture is constructed. The difference of the control structure will be considered in the continuation of this project.

### A.6.6 Mixed Control with Partially-Distributed Computer

This is a natural combination were the mixed control, Figure A.2, and the partially-distributed computer architecture, Figure A.5, match. With this combination the calculations for the local control loops are made at the distributed nodes and relieve the central node of computation.

The cabin node is assumed to handle the case were the central node fails, as in section A.6.3. This is done by reconfiguration and handling of the critical functionality, i.e., the basic brake functionality, where the brake pressure actuators shall be provided with set values from the cabin node instead of the central node.

In this system all information is no longer available on the bus. However, some messages are assumed to be needed by other nodes. These messages have been indicated with "node ext.". This combination also introduce a new set of messages. They arise since the control is divided into a central controller and a number of local controllers. The data in the interface between them is sent via the bus, see Figure A.2. From the central controller, data are sent as set points for the local controllers located in the distributed nodes. In the other direction, necessary sensor data are sent.

#### Simplex System

In Table A.20 the bandwidth for the simplex system is calculated. Node internal data are indicated with a zero in the "bandwidth" column.

In this table can we see that data sent between the central controller and the local controllers are similar to the central strategy case in Table A.17.

#### Duplex System

In the duplex system only the cabin node with the pedal sensors are duplicated since it will handle the critical functionality if the central node fails. In order to keep the duplicated nodes in the same state and ensure that they send the same data to nodes they will agree on transmitted information. This is handled the same way as with the central control together with a partially-distributed architecture, see section A.6.3.

In Table A.21 the bandwidth for the duplex system is calculated. The bandwidth requirements for most nodes are similar to the simplex case and therefore only changes are specified.

In this duplex configuration, we have a considerable increase in required data exchange compared to the simplex case, due to the duplex

| Sensors | bits | # | Tot. | Hz | Bits/sec |
|---|---|---|---|---|---|
| **Central node** | | | | | |
| Vehicle speed, (Virtual sensor) | 16 | 1 | 16 | 50 | 800 |
| Engine torque setpoint | 16 | 1 | 16 | 20 | 320 |
| Retarder setpoint | 16 | 1 | 16 | 20 | 320 |
| Engine brake setpoint | 16 | 1 | 16 | 100 | 1600 |
| Brake pressure setpoint x 6 | 16 | 6 | 96 | 100 | 9600 |
| Rear wheel steering setpoint | 16 | 6 | 96 | 20 | 1920 |
| **Total number bits** | | | | | *14560* |
| **Cabin node** | | | | | |
| Brake pedal sensor | 16 | 1 | 16 | 100 | 1600 |
| Steering wheel sensor | 16 | 1 | 16 | 20 | 320 |
| Accelerator pedal sensor | 16 | 1 | 16 | 20 | 320 |
| Gear selection sensor | 8 | 1 | 8 | 10 | 80 |
| **Total number bits** | | | | | *2320* |
| **Sensor node** | | | | | |
| Sensor for lateral acceleration | 16 | 1 | 16 | 10 | 160 |
| Yaw rate sensors | 16 | 1 | 16 | 10 | 160 |
| Articulation angle | 16 | 1 | 16 | 10 | 160 |
| **Total number bits** | | | | | *480* |
| **Engine node** | | | | | |
| Engine Torque sensor (node int.) | 16 | 1 | 16 | 100 | 0 |
| Engine Torque sensor (node ext.) | 16 | 1 | 16 | 10 | 160 |
| Engine rpm sensor (node int.) | 16 | 1 | 16 | 100 | 0 |
| Engine rpm sensor (node ext.) | 16 | 1 | 16 | 10 | 160 |
| Engine torque actuator | 16 | 1 | 16 | 100 | 0 |
| Retarder actuator (Hydraulic) | 16 | 1 | 16 | 100 | 0 |
| Retarder actuator (Electro magnetic) | 16 | 1 | 16 | 100 | 0 |
| Engine brake actuator | 16 | 1 | 16 | 100 | 0 |
| **Total number of sensor data** | | | | | *320* |
| **Wheel nodes x 6** | | | | | |
| Wheel speed sensors | 16 | 6 | 96 | 50 | 4800 |
| Brake pressure actuator | 16 | 1 | 16 | 100 | 0 |
| Rear Wheel steering actuator x 4 (four RW) | 16 | 4 | 64 | 20 | 0 |
| Trailer brake pressure sensor* | 16 | 1 | 16 | 50 | 800 |
| Trailer brake pressure actuator x 1 | 16 | 1 | 16 | 100 | 0 |
| **Total number of sensor data** | | | | | *5600* |
| **Total required bandwidth** | | | | | **23280** |
| **Max frequency** | | | | **100** | |

Figure A.20: Bandwidth requirement configuration with Mixed Control and Partially-Distributer computer architecture

cabin node and the consistency requirement among the replicas of the cabin node. It is though considerably less than for the redundant central system A.6.2. For this configuration the effects of the duplex nodes and consistency requirements are not as big since there are not that much data that must be agreed upon.

| Sensors | bits | # | Tot. | Hz | Bits/sec |
|---|---|---|---|---|---|
| **Cabin node x 2 (RD messages)** | | | | | |
| Brake pedal sensor | 16 | 4 | 64 | 100 | 6400 |
| Steering wheel sensor | 16 | 4 | 64 | 20 | 1280 |
| Accelerator pedal sensor | 16 | 4 | 64 | 20 | 1280 |
| Gear selection sensor | 8 | 4 | 32 | 10 | 320 |
| **Total number bits** | | | | | ***9280*** |
| **Central node** | | | | | 14560 |
| **Sensor node** | | | | | 480 |
| **Engine node** | | | | | 320 |
| **Wheel nodes x 6** | | | | | 5600 |
| **Total required bandwidth** | | | | | **30240** |
| **Max frequency** | | | | **100** | |

Figure A.21: Bandwidth requirement, configuration with Mixed Control and Partially-Distributer computer architecture and redundancy.

### Fault Scenarios

With this situation we have a similar situation as with the configuration with central control and a partially-distributed architecture A.6.3. The cabin node will have the possibility to reconfigure and run critical functionality in case of a failure of the central node. However, the design complexity for achieving this will be less compared to the central control case (with partially distributed computers), since the mixed control already have distributed some functionality that can be "reused". The complexity of the software part of the system can thereby be reduced compared to the configuration with central control and a partially-distributed architecture. The outage time requirement will not be changed. This will require the same fast error detection of a failed central node.

Considering the duplex configuration, we have lower availability of the total functionality compared to a central configuration with duplicated nodes, since the central node is simplex in this configuration. Furthermore, we have the same requirement as with the central-control/partially-distributed configurations considering fail-silent cabin-nodes, and that the other nodes should be fail-safe.

The reintegration cost of state exchange between replicas in the duplex system is assumed to be less than the centralized-control/central architecture configuration. The necessary information exchange between the replicas is the state information for the local loops of the mixed control.

It is likely that a mixed-control solution will provide the system with a higher robustness to transient failures. When the local loops are spread to the distributed-nodes, the local loops have a better possibility to handle situations, such as lost samples, compared to a central system. This can

be done by estimating the lost sample based on the application knowledge. This functionality is normally included in the local loops at control design. Similar behavior could be used in a system with central control, but additional complexity would be needed at the "intelligent" sensors/actuators. This is also an area for further studies.

### A.6.7 Mixed Control with Fully-Distributed Computer

This combination may be interesting. With this solution we assume that the central controller is distributed to all nodes. This means that each node runs an own copy of the central controller in addition to the local controller. The least necessary information transfer will be the same as for a configuration with mixed-control/partially-distributed architecture. Since this configuration has no central node, it would require one of the distributed nodes to send the required set-points.

The big advantage of this solution would be if each node distributes its copy of the set points for the local controllers. This results in a high degree of redundancy but will increase the bandwidth significantly. This is also the solution chosen in [ATJ99]. However, compared to a Flight Control system, a truck is a fairly flexible system where additional nodes should be possible to add. This combined with the increased bandwidth required, makes us consider this combination unsuitable for our purposes.

### A.6.8 Local Control with Central Computer

This combination is not meaningful, since the concept of one central computer node dose not fit the concept of a number of local controllers.

### A.6.9 Local Control with Partially-Distributed Computer

See section A.6.8.

### A.6.10 Local Control with Fully-Distributed Computer

In the last combination we have a number of local controllers running on a number of distributed nodes. The local controllers only exchange necessary information. In this case there is no central node. The cabin node is the critical node and thus duplicated, since it has the pedal sensors connected.

#### Simplex System

In Table A.22 the bandwidth for the simplex system is calculated. Node internal data are indicated with a zero in the "bandwidth" column. The

table shows that the bandwidth required is considerably less compared to the central-control/centralized-computer combination. In this configuration a number of sensor values are not necessary to be transmitted. A number of sensor values have been divided into node internal and node external. Node internal data are used for fast internal loops. Node external data can be transferred with a lower frequency. The bandwidth difference between this configuration and the mixed-control/partially-distributed configuration comes from the fact that in the later configuration set-point values are sent from the central-node/mixed-controller.

| **Sensors** | **bits** | **#** | **Tot.** | **Hz** | **Bits/sec** |
|---|---|---|---|---|---|
| **Cabin node** | | | | | |
| Brake pedal sensor | 16 | 1 | 16 | 100 | 1600 |
| Steering wheel sensor | 16 | 1 | 16 | 20 | 320 |
| Accelerator pedal sensor | 16 | 1 | 16 | 20 | 320 |
| Gear selection sensor | 8 | 1 | 8 | 10 | 80 |
| **Total number bits** | | | | | *2320* |
| **Sensor node** | | | | | |
| Sensor for lateral acceleration | 16 | 1 | 16 | 10 | 160 |
| Yaw rate sensors | 16 | 1 | 16 | 10 | 160 |
| Articulation angle | 16 | 1 | 16 | 10 | 160 |
| **Total number bits** | | | | | *480* |
| **Engin node** | | | | | |
| Engine Torque sensor (node int.) | 16 | 1 | 16 | 100 | 0 |
| Engine Torque sensor (node ext.) | 16 | 1 | 16 | 10 | 160 |
| Engine rpm sensor (node int.) | 16 | 1 | 16 | 100 | 0 |
| Engine rpm sensor (node ext.) | 16 | 1 | 16 | 10 | 160 |
| Engine torque actuator | 16 | 1 | 16 | 100 | 0 |
| Retarder actuator (Hydraulic) | 16 | 1 | 16 | 100 | 0 |
| Retarder actuator (Electro magnetic) | 16 | 1 | 16 | 100 | 0 |
| Engine brake actuator | 16 | 1 | 16 | 100 | 0 |
| **Total number of sensor data** | | | | | *320* |
| **Wheel nodes x 6** | | | | | |
| Wheel speed sensors | 16 | 6 | 96 | 50 | 4800 |
| Brake pressure actuator | 16 | 1 | 16 | 100 | 0 |
| Rear Wheel steering actuator x 4 (four RW) | 16 | 4 | 64 | 20 | 0 |
| Trailer brake pressure sensor* | 16 | 1 | 16 | 50 | 800 |
| Trailer brake pressure actuator x 1 | 16 | 1 | 16 | 100 | 0 |
| **Total number of sensor data** | | | | | *5600* |
| **Total required bandwidth** | | | | | **8720** |
| **Max frequency** | | | | **100** | |

Figure A.22: Bandwidth requirement, simplex configuration with Local Control together with a Fully-Distributed computer architecture.

### Duplex System

In Table A.23 the bandwidth is calculated for the configuration with local control and a fully-distributed architecture. In the table we see that this configuration has a very low bandwidth requirement since a minimal amount of data must be transmitted. The information exchange needed to ensure consistency also becomes low for the same reason. As in the simplex case, the bandwidth difference between this configuration and the mixed-control/partially-distributed configuration comes from the fact that in the later configuration set-point values are sent from the central-node/mixed-controller.

| Sensors | bits | # | Hz | Bits/sec |
|---|---|---|---|---|
| **Cabin node x 2 (RD messages)** | | | | |
| Brake pedal sensor | 16 | 4 | 100 | 6400 |
| Steering wheel sensor | 16 | 4 | 20 | 1280 |
| Accelerator pedal sensor | 16 | 4 | 20 | 1280 |
| Gear selection sensor | 8 | 4 | 10 | 320 |
| **Total number bits** | | | | ***9280*** |
| **Sensor node** | | | | 480 |
| **Engin node** | | | | 320 |
| **Wheel nodes x 6** | | | | 5600 |
| **Total required bandwidth** | | | | **15200** |
| **Max frequency** | | | **100** | |

Figure A.23: Bandwidth requirement, configuration with Local Control together with a Distributer computer architecture and a duplex node.

### Fault Scenarios

This configuration has no central node. Thus, the total functionality can be more robust, since the loss of one node might only affect a limited part of the total functionality. In a simplex configuration the cabin node is vital for the critical functionality, since it has the pedal sensor. To avoid this single point of failure, it is recommendable to use a duplicated cabin node. Together with a duplex bus this will avoid a single point of error.

This solution will not need any reconfiguration in case of a failure. The working nodes will continue as normal. This requires that the receivers of data from the duplicated node, i.e., the replicas, can distinguish data from correct and faulty replicas. This reintroduces the requirement of fail-silence on the replicas of the cabin node. The requirement, on the other nodes, of fail safe behavior is still valid.

We have reduced the complexity of system software considerably when reconfiguration can be removed. For reconfiguration we need at least two

software parts: The first when working under normal conditions and the other after the reconfiguration. The reconfiguration is critical since it comes in a situation where something already has gone wrong. Considerable efforts must be spent to ensure that the reconfiguration "always" will work, and similarly with the software that run after a reconfiguration. It is therefore beneficial to avoid reconfigurations and to run only a single well verified program.

The local-controllers will most probably be more complex than controllers based on centralized or mixed control architecture. This is however a question for further investigation.

The handling of transient errors is robust as for the system with mixed control. With local control each node has a greater "knowledge" of the system behavior, which can be used for better estimation of for example lost samples.

## A.7 Summary and Conclusions

In this chapter we will summarize and conclude our reliability analysis and bandwidth study. We start by giving a short summary of the different configurations and their main features in Table A.8.

In Table A.3 a summary of the reliability results can be seen. Here we can conclude that we have a small difference in the reliability concerning the total yaw rate control function. The centralized system has a little higher reliability of the total functionality. The advantage of the central architecture comes from the fact that it only has one complex node and a number very simple "sensor nodes". If the complexity of the distributed nodes is decreased the differences between the central and the fully-distributed solutions will decrease.

When only considering the critical functionality the partially-distributed architecture has the highest reliability. The gap to the fully distributed architecture is however not big. The difference comes from the higher complexity of the nodes in a fully-distributed system. In the central system the complexity of the central node is the reason for its less reliable behavior.

From the sensitivity analysis, we can note the value of keeping the node complexity low, especially nodes that are critical and will be duplicated. This is an incentive to keep complexity low on nodes that will be redundant.

Another important factor to consider, is the effect of the error detection coverage. With a high coverage there is a small probability that one replica of a redundant node will fail in such a way that it brings the system down. This can happen if, e.g, the fail silent property is violated

| Control | **Centralized** | **Partially-Dist.** | **Fully-Dist.** |
|---|---|---|---|
| **Central** | Low reliability. Sensor and actuator data sent on bus. High communication bandwidth. | Low reliability. A distributed node can handle critical functionality after reconfiguration. Medium communication requirement. | Not considered, since close conceptually to local-control/fully-Distributed configuration. |
| **Mixed** | Not considered. Only node internal difference from central-control/centralized config. | Low reliability. Medium communication requirement. | Not considered. |
| **Local** | Not considered. | Not considered. | Medium reliability. Low communication requirement. No reconfiguration necessary. |

Table A.8: Short summary of configurations.

and faulty data is propagating throughout the system. Table A.3 clearly show the effect and importance of the coverage for the duplex system.

The bandwidth requirements are summarized in Figure A.24. The combination with local control and a fully-distributed architecture clearly has the lowest communication bandwidth requirements, both for a simplex and duplex system. Indication of low bandwidth requirements for a fully-distributed solution can be found in other reports, such as for example [Lön99b]. This combination clearly minimizes the necessary information exchange and most likely also the overhead to ensure consistency. The reason for this is the local control that can handle all local loops and at the same time minimize sensor value exchange between nodes.

The configuration with central control and a central architecture is the combination which require most information exchange. The central architecture requires all sensor and actuator data to be transferred to/from the computer node, see Figure A.7.

| Combination | Simplex | Duplex |
|---|---|---|
| Central Control with a Central Arch. | *23280* | *57680* |
| Central Control with Half Dist. Arch. | *23280* | *29440* |
| Mixed Control with Half Dist. Arch. | *23280* | *30240* |
| Autonomous Control with a Dist. Arch . | *8720* | *15680* |

Figure A.24: Bandwidth requirement, for the different configurations.

A short discussion about fault scenarios for the different configurations is included in this report. They will not be summarized here, but a few important comments will be made. There is a cost associated with the re-integration of a replica in a duplex configuration. This will affect the possibility to schedule the communication since "integration state" data must be sent within a reasonable time. The communication schedule must consider this and reserve enough communication bandwidth such that re-integration is possible. The necessary state information transfer is dependent on the application. Most probably the necessary state transfer is related to the complexity of the applications running on a node, a central control with a central architecture will require a lot of information exchange at reintegration. This will further add to this configurations high communication bandwidth. The partially-distributed and distributed systems will have the lower communication overhead for reintegration for each node since they will contain smaller state. Since the nodes that are duplicated in the partially-distributed and distributed systems contains a smaller state (they are less complex) compared to the central node they will require a smaller state transfer in case of reinte-

gration. This do not include the configuration with central control and a partially-distributed architecture where almost no work is done at the "distributed" nodes.

The membership agreement service has been discussed and is necessary in for example the partially-distributed system. However, this service will probably be required by the control application, e.g., to adapt to the loss of wheel brake actuator. This is an area of further research, and will be investigated in the control-system report.

One drawback with the partially-distributed architecture is the reconfiguration, which is necessary if the central node fails. This adds to the complexity both during runtime and during scheduling. This reconfiguration complexity could be traded with an duplex central node, which instead would increase hardware and communication cost.

The robustness to transient faults is an important issue and may be affected by the control strategy. The distributed nodes of a mixed-control/partially-distributed architecture have more application information. This knowledge can be used when a transient error occur, to tolerate or estimate the correct state. One example can be a lost sample which can be estimated and thus tolerated. The local control/distributed architecture has even more application information to use. How good this could be will need further investigation, and will to some extent be studied within the continuation of the project.

Considering all the facts so far, we have two configurations that we consider very applicable in this type of distributed control, (1) a duplex system with a configuration with local-control and a fully-distributed computer architecture, and (2) a duplex configuration with mixed-control and a partially-distributed computer architecture. Even though the communication bandwidth is considerable less in local-control/fully-distributed, the final implementation may benefit the mixed-control/partially-distributed configuration. The intention is to further evaluate these configuration when the control design is finished and can be put in these environments. Below follows a summary of the main benefits and reasons to choose these configurations:

**Duplex** A duplicated system will be necessary to achieve sufficient reliability and to avoid single point of failures. Even though we can not trust the exact figures of the reliability analysis, they give a strong hint that a simplex system is not enough. Furthermore, a single point of error is hardly tolerated.

**local-control/fully-distributed** 1. This combination has the lowest bandwidth by far of all duplex systems.

2. The fully-distributed architecture does not require reconfiguration after a fault and it has only slightly lower reliability than the partially-distributed architecture. Furthermore, each node have application knowledge which can be used to tolerate transient faults, e.g., lost samples.

**mixed-control/partially-distributed**

1. This configuration has the highest reliability figures.
2. The development of mixed-control is closer to existing and well known control development methods used today, which results in better chances of reuse of both control applications and development methods.

These configurations will be further investigated and evaluated within this project when more is known about the control solution.

# Appendix B

# Appendix B

## B.1 Abbreviations

- AYC - Active yaw-control.
- ARC - Active roll-control.
- ABS - Anti-locking brake system.
- TRC - Traction control, other existing abbreviations are ATC and TCS.
- VDC - Vehicle dynamics control, consists of AYC, ARC, ABS, and TRC.
- RWS - Rear wheel steering.
- EBS - Electronic brake system, consists of electronically controlled ABS and TRC.
- ESP - Electronic stability program, relies on EBS, and implements VDC. Please note the difference in our interpretation of VDC and ESP.
- EMS - Electronic engine system.
- ECS - Electronic air cushion system. Controls the heigth of the vehicle.
- TCE - Trailer control ECU. A gateway to the semi-trailer.
- ROP - Roll over protection.
- VECE - Main vehicle computer.

- Total functionality - Refers to a fully functioning yaw-control system.
- Critical functionality - Refers to a degraded mode where at least the basic brakes are working.

## B.2 Failure Rates

In this part the failure rates of the different components are derived from the MIL-HDBK-217 standard. The different components are listed and data used to extract the failure rate for each specific component are described. Below follows a short description of the different components in the failure rate calculations. Further information can be found in MIL-HNDBK [MIL92].

A computer node will consist of the components in fig B.1, which are the integrated circuits for power supply, bus interface, bus driver and the microprocessor. This picture also describes the internal components of sensors and actuators connected directly to the bus.

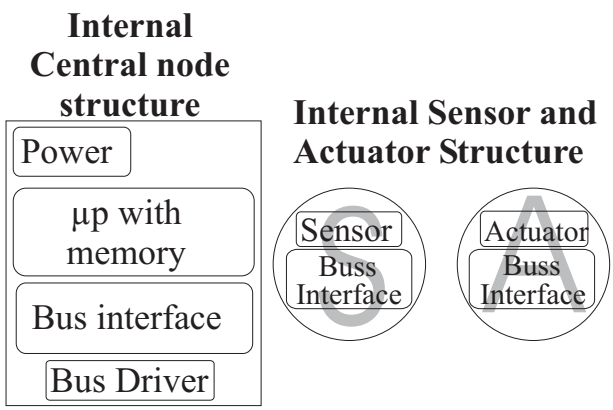


Figure B.1: The internal components of computer nodes and sensors and actuators connected to the bus.

The microprocessors will be a bit different depending on the computer architecture selected. A central node handling all calculations must be more powerful than a node in a distributed system where computations are spread among a number of nodes. Thus we have three different microprocessors for which we calculate the failure-rate. The difference will be the complexity of the chip.

This list will give a short description of the constants which will be used in the failure rate calculation.

**C1** is a complexity factor, dependent on complexity of the integrated circuit.

$\pi_T$ is a temperature factor, dependent on semiconductor process and working temperature. It also dependent of the maximum junction temperature $T_j$.

**C2** is a complexity factor dependent on the complexity of the packaging.

$\pi_E$ is a environmental factor dependent on the components working environment.

$\pi_Q$ is the quality factor, and is dependent on the amount of testing done on the component after production, i.e., the screening level.

$\pi_L$ is the learning factor, indicates the confidence in the manufacturing process. Components from new and unproven manufacturing processes are given low confides, i.e., $\pi_L = 10$, otherwise $\pi_L = 1$. We assume that components used in these type of application will use standard components, which have been in production for a while. Thus $\pi_L = 1$ are assumed.

### B.2.1 Microprocessor in a Central Node

The failure rate is derived from the formula:

$$\lambda_{\mu P} = (C1\,\pi_T + C2\,\pi_e)\;\pi_L\,\pi_Q \times 10^{-6} = 1.0 \times 10^{-5} failures/hour \quad \text{(B.1)}$$

The following values on the constants have been used.

**C1** = 0.56 (a 32 bits Mos processor)

$\pi_T$ = 1.1 (Digital, Mos, $T_j = 90^\circ$ C)

**C2** = 0.097 (for 224 pins, hermetic DIPs)

$\pi_E$ = 4.0 (Environmental factor, $G_M$ (Ground mobile))

$\pi_Q$ = 10 (Other commercial or unknown screening levels)

$\pi_L$ = 1 (Learning factor)

### B.2.2 Microprocessor in Partially-Distributed Nodes

The failure rate is derived from the formula:

$$\lambda_{\mu P2} = (C1\,\pi_T + C2\,\pi_e)\;\pi_L\,\pi_Q \times 10^{-6} = 2.82 \times 10^{-6} failures/hour \quad \text{(B.2)}$$

The following values on the constants have been used.

**C1** = 0.14 (16 bits, Mos processor)

$\pi_T$ = 1.1 (Digital, Mos, $T_j = 90^\circ$ C)

**C2** = 0.032 (for 80 pins, hermetic DIPs)

$\pi_E$ = 4.0 (Environmental factor, $G_M$ (Ground mobile))

$\pi_Q$ = 10 (Other commercial or unknown screening levels)

$\pi_L$ = 1 (Learning factor)

### B.2.3 Microprocessor in Fully-Distributed Nodes

The failure rate is derived from the formula:

$$\lambda_{\mu P3} = (C1\,\pi_T + C2\,\pi_e)\;\pi_L\,\pi_Q \times 10^{-6} = 5.2 \times 10^{-6} failures/hour \quad \text{(B.3)}$$

The following values on the constants have been used.

**C1** = 0.28 (a 16 bits Mos processor)

$\pi_T$ = 1.1 (Digital, Mos, $T_j = 90^\circ$ C)

**C2** = 0.053 (for 128 pins, hermetic DIPs)

$\pi_E$ = 4.0 (Environmental factor, $G_M$ (Ground mobile))

$\pi_Q$ = 10 (Other commercial or unknown screening levels)

$\pi_L$ = 1 (Learning factor)

### B.2.4 Communication Interface

The failure rate is derived from the formula:

$$\lambda_{CI} = (C1\,\pi_T + C2\,\pi_e)\;\pi_L\,\pi_Q \times 10^{-6} = 3.48 \times 10^{-6} failures/hour \quad \text{(B.4)}$$

The following values on the constants have been used.

**C1** = 0.2 (Mos, 30 000 gates, Digital)

$\pi_T$ = 1.1 (Digital, Mos, $T_j = 90^\circ$ C)

**C2** = 0.032 (for 80 pins, hermetic DIPs)

$\pi_E$ = 4.0 (Environmental factor, $G_M$ (Ground mobile))

$\pi_Q$ = 10 (Other commercial or unknown screening levels)

$\pi_L$ = 1 (Learning factor)

### B.2.5 Power IC

The failure rate is derived from the formula:

$$\lambda_{PIC} = (C1\,\pi_T + C2\,\pi_e)\;\pi_L\,\pi_Q \times 10^{-6} = 2.21 \times 10^{-6} failures/hour \tag{B.5}$$

The following values on the constants have been used.

**C1** = 0.02 (Mos, 300 gates, Linear)

$\pi_T$ = 9.3 (Linear, Mos, $T_j = 90^\circ$ C)

**C2** = 0.0087 (for 24 pins, hermetic DIPs)

$\pi_E$ = 4.0 (Environmental factor, $G_M$ (Ground mobile))

$\pi_Q$ = 10 (Other commercial or unknown screening levels)

$\pi_L$ = 1 (Learning factor)

### B.2.6 Bus Driver

The failure rate is derived from the formula:

$$\lambda_{BD} = (C1\,\pi_T + C2\,\pi_e)\;\pi_L\,\pi_Q \times 10^{-6} = 1.1 \times 10^{-6} failures/hour \tag{B.6}$$

The following values on the constants have been used.

**C1** = 0.01 (Mos,Linear)

$\pi_T$ = 9.3 (Linear, Mos, $T_j = 90^\circ$ C)

**C2** = 0.0041 (for 24 pins, hermetic DIPs)

$\pi_E$ = 4.0 (Environmental factor, $G_M$ (Ground mobile))

$\pi_Q$ = 10 (Other commercial or unknown screening levels)

$\pi_L$ = 1 (Learning factor)

### B.2.7 Bus Connections

For the failure rate of the bus connection, the least expensive materials are assumed. Insert Material D, ambient temperature $90^\circ$ C for ($\lambda_b$). Matining/unmating factor 0 to 0.05 per 1000h (for $\lambda_k$). Aktive pinfactor is 6 (for $\pi_p$) Environmental factor is $G_M$ (for ground vehicles), not military standard (for $\pi_E$).

$\lambda_b$ = 0.033

$\lambda_k = 1.0$

$\pi_p = 2.0$

$\pi_E = 21$

$$\lambda_{con} = \pi_e \, \pi_p \, \lambda_b \, \lambda_k \times 10^{-6} = 1.39 \times 10^{-6} \tag{B.7}$$

Normally a failure in the bus connection only affects the one node, i.e., the node where the failed bus connector is attached. However, in rare cases it may affect the whole bus in which case it will make the bus useless for all nodes. MIL-HNDBK does not give any figures for this but it has been estimated that the failure rate is a factor 10 less than the normal bus connection failure rate a. This is probably a pessimistic estimate. This means that the failure rate for a bus is related to the existent number of connection to the bus. This gives us for a system with n nodes:

$$\lambda_{bus} = \frac{n \, \lambda_{con}}{10}$$

### B.2.8 Sensors and Actuators

The MIL-HNDBK does not cover failure rates for sensors and actuators since they are too specific in nature. Thus, we have been forced to make assumptions about the failure rate for the sensors and actuators. However, in order to see the impact of these failure rate most of the calculations have been made with failure rates from as low as $10^{-6}$ to very high $100 \times 10^{-6}$.

The failure rate for sensors $\lambda_s$ are assumed to be the same as the failure rate for the actuators $\lambda_a$ through out this report, i.e., $\lambda_s = \lambda_a$

### B.2.9 Sensor and Actuator Communication Interface

The failure rate are derived from the formula:

$$\lambda_{bCI} = (C1 \, \pi_T + C2 \, \pi_e) \; \pi_L \, \pi_Q \times 10^{-6} = 1.48 \times 10^{-6} \tag{B.8}$$

The following values on the constants have been used.

**C1** = 0.08 (Mos, 10 000 gates, Digital)

$\pi_T$ = 1.1 (Digital, Mos, $T_j = 90^{\circ}$ C)

**C2** = 0.015 (for 40 pins, hermetic DIPs)

$\pi_E$ = 4.0 (Environmental factor, $G_M$ (Ground mobile))

$\pi_Q = 10$ (Other commercial or unknown screening levels)

$\pi_L = 1$ (Learning factor)

## B.3 Reliability Calculations

### B.3.1 Central System

#### Simplex Configuration

**The Central node**

The failure rate and reliability for the computer node in the central system is shown in (B.9) and (B.10), where the different values can be found in Appendix B.2. The configuration of the node is shown in Figure B.1, it consist of a microprocessor, power unit, bus interface and a bus driver.

$$\lambda_{simplexnode} = \lambda_{BD} + \lambda_{CI} + \lambda_{PIC} + \lambda_{\mu P} + \lambda_{con} = 1.82 \times 10^{-5} \quad \text{(B.9)}$$

$$R_{Cnode} = e^{-t\left(\lambda_{BD}+\lambda_{CI}+\lambda_{PIC}+\lambda_{\mu P}+\lambda_{con}\right)} \quad \text{(B.10)}$$

The reliability will be calculated for an hour throughout this report. Then we get the reliability of the Central node to:

$$R_{Cnode} = 0.999982$$

This reliability also considers bus failures. However, it does not include bus failures that affect other node's possibility to communicate on the bus. Failures that eliminate bus communication will be treated in the next section.

**The Bus**

In this part we will estimate reliability of the bus connections. More specifically we consider failure mode that affect the whole bus. For example short circuit of the bus, which prohibits any bus communication. Since the unreliability originates from the bus connection, the reliability will be dependent on the number of connections on the bus.

The MIL-HNDBK does not support this division of the reliability of connectors. Therefor, the failure rate of connections that affects the whole bus will be estimated. $\lambda_{con}$ is the failure rate of one connection. The failure rate of the part affecting the whole bus, $\lambda_{bus}$, is estimated to one tenth of $\lambda_{con}$. See also section B.2.7.

The bus will have 17 units connected to it. Further more we assume that in this system an additional number of nodes will be connected to the same bus increasing the number of connection points to 40, i.e., additional

23 connections. This will give us the number of bus connections $n = 40$. This will result in failure rate of:

$$\lambda_{bus} = \frac{n\,\lambda_{con}}{10}$$

The reliability for the bus in the central system is $R_{bus}$:

$$R_{bus} = e^{-\lambda_{bus}} = 0.9999945$$

### Sensors and Actuators

The failure rate for sensors and actuators is the same for all nodes and is obtained from the sum of the failure rate from the sensor/actuator and the bus connector. The failure fate and reliability of a sensor or an actuator is thus:

$$\lambda_{sens} = \lambda_s + \lambda_{bCI} + \lambda_{con} + \lambda_{BD} \tag{B.11}$$

$$R_s = R_a = e^{-\lambda_{sens}} \tag{B.12}$$

### Probability of System Failure for the Total Functionality of the Central System

The probability of a system function with one central node, ten sensors and six actuators is calculated here. These parts can be considered as a series system, and we get the reliability as:

$$R_{cs} = R_{Cnode}\,R_{bus}\,R_s^{10}\,R_a^6 = 0.999976 \times e^{-\left(6.3\times 10^{-5}+16\,\lambda_s\right)}$$

The probability of loss of any system functionality is thus:

$$Q_{CS} = 1 - R_{CS}$$

In Figure B.2 the probability of losing functionality in our yaw-control system is plotted. The variable is the failure-rate of the sensors and actuators, i.e., $\lambda_s$ and $\lambda_a$ the failure

### Probability of System Failure for the Critical Functionality

Here the probability of loss critical parts of the system function will be calculated. The critical parts that are required are the central node and brake pedal sensors and two brake actuators. One brake actuator on each side of the truck must be functional for the system to be considered working, i.e., have at least minimal braking performance. Thus the reliability

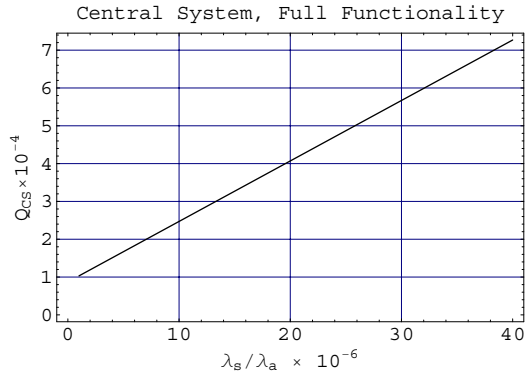


Figure B.2: The probability for losing any functionality of the central system.

of brake nodes one side is, i.e., the probability that one of three brake actuator is working:

$$R_{oneside} = 1 - (1 - R_a)^3 \tag{B.13}$$

$R_a$ is the brake actuator (see equation (B.12) for explanation). Then the reliability for the critical parts of the system can be calculated. The critical parts are the central node, bus, brake pedal sensor, and at least one actuator node at each side. Thus we get the reliability $RC_s$, and the probability of critical failure $QC_{cs}$ .

$$RC_{cs} = R_{Cnode}\, R_{bus}\, R_s\, (R_{oneside})^2$$

$$QC_{cs} = 1 - RC_{cs} = 1 - 0.999972 \times e^{\lambda_s} \left( \left( 1 - 0.999996 \times e^{\lambda_s} \right)^3 - 1 \right)^2$$

In Figure B.3 the probability of losing critical functionality per hour is plotted. As variable the failure rate for sensors and actuators are varied from $10^{-6}$ to $40 \times 10^{-6}$.

### Redundant System

By study the reliability of "one brake on each side working" we find that we get a very high reliability for that part, e.g., $5 \times 10^{-15}$ for $\lambda_s = 10 \times 10^{-6}$. Then we can conclude that the brake pressure actuators have little impact on the system unreliability, since they have an inherent redundancy. However, we can easily improve the system reliability by duplication of the central node, the bus and the brake-pedal sensor. For the central node we have to consider what will happen if one

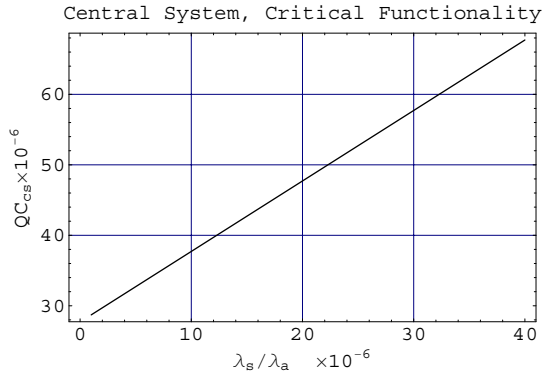


Figure B.3: The probability of losing the critical functionality of the central system.

node fails. A node is considered to fail in two ways, (1) the node fails but will not disturb the operation of the fault free part of the system. (2) The nod fails in such a way that required functionality is not upheld. To achieve Fault containment or Fail-Silence the system must be able to detect and handle the faults, e.g., be silent in the case of a fault. The coverage of the necessary error detection mechanisms is important and will affect the system reliability. In this system we assume that we have a coverage (C) between 0.99 and 0.999. This is high but we will by varying C to see the importance of the coverage C. Considerable efforts must be used to ensure a high coverage.

By Markov modeling we get the following result. $P_2\,(t)$ the probability for being in a state where both nodes work. $P_1\,(t)$ is the probability of being in a state where one node is faulty and finally $P_F\,(t)$ is the probability of a system failure. C is the coverage.

$$\begin{aligned} P_2'(t) = &\quad -2\,\lambda\,P_2(t) \\ P_1'(t) = &\quad 2\,C\,\lambda\,P_2(t) - (\lambda\,P_1(t)) \\ P_F'(t) = &\quad \lambda\,P_1(t) + 2\,(1-C)\lambda P_2(t) \end{aligned}$$

Thus, the equation system in (B.14) follows.

$$\begin{pmatrix} P_2'(t) \\ P_1'(t) \\ P_F'(t) \end{pmatrix} = \begin{pmatrix} -2\,\lambda & 0 & 0 \\ 2\,C\,\lambda & -\lambda & 0 \\ 2\,(1-C)\lambda & \lambda & 0 \end{pmatrix} \cdot \begin{pmatrix} P_2(t) \\ P_1(t) \\ P_F(t) \end{pmatrix} \tag{B.14}$$

To solve this differential equation system we Laplace transform the equation system. After which we get

$$\begin{pmatrix} P_2(s) \\ P_1(s) \\ P_F(s) \end{pmatrix} = \begin{pmatrix} \frac{1}{s+2\,\lambda} \\ \frac{2\,C}{s+\lambda} - \frac{2\,C}{s+2\,\lambda} \\ \frac{1}{s} - \frac{2\,C}{s+\lambda} + \frac{2\,C-1}{s+2\,\lambda} \end{pmatrix} \tag{B.15}$$

Inverse transform of (B.15) gives:

$$\begin{pmatrix} P_2(t) \\ P_1(t) \\ P_F(t) \end{pmatrix} = \begin{pmatrix} e^{-2\,t\,\lambda} \\ -2\,C\,e^{-2\,t\,\lambda} + 2\,C\,e^{-t\,\lambda} \\ 1 + (2\,C-1)\,e^{-2\,t\,\lambda} + 2\,C\,e^{-t\,\lambda} \end{pmatrix} \tag{B.16}$$

Thus the probability for loss of the central node is $P_F(t)$. Which we get using $\lambda_{simplexnode}$ from equation (B.9). This gives a reliability of the central nodes for one hour of $R_{dupCnode} = 1 + P_F(1)$. Then the reliability is:

$$R_{dupCnode} = 0.999963 + C\,3.64\ \times 10^{-5}$$

Where C is the coverage of the fault detection.

We will here assume that we use three pedal sensors. For these three sensors we assume that they do not fail in a way that affect the other sensors. However, we assume that we need two of the three sensors for correct operation. The reliability of a 2 - of - 3 system is (with independent components with the same reliability R) $3\,R^3 - 2\,R^2$. For one sensor the failure rate was calculated in (B.11), that is $\lambda_{sens}$ and the reliability from (B.12). Then we get the reliability of the Pedal in equation

$$R_{dupPS} = 3\,R^3 - 2\,R^2 = 2.999983\,e^{-2\,\lambda_s} - 1.999983 * e^{3\,\lambda_s} \tag{B.17}$$

The reliability of the double bus is:

$$R_{dupBus} = 1 - (1 - R_{bus})^2$$

The reliability for the brake pressure nodes, where one of three node on each side is required to work, is:

$$R_{dupBP} = \left(1 - (1 - R_a)^3\right)^2$$

For the brake pressure actuators we have the same as in equation (B.13), that is $\lambda^2_{oneside}$. Then the reliability for the central system, the pedal sensors, the double bus, and the brake pressure actuators are used to calculate the reliability of the critical part of the system.

$$R_{dupC} = R_{dupCnode}\,R_{dupPS}\,R_{RdupBus}\,R_{dupBP} \tag{B.18}$$

The probability for system faulure per hour is thus:

$$Q_{dup} = 1 - R_{dupC} = \left(0.999966 + 3.47 \times 10^{-5}\, C\right) \left(3\, e^{-5.73\times10^{-6}-2\,\lambda_s} - 2\, e^{-8.60\times10^{-6}-3\,\lambda_s}\right) \left(1 - \left(1 - e^{-2.87\times10^{-6}-\lambda_s}\right)^3\right)^2 \quad \text{(B.19)}$$

Where $\lambda_s$ is the failure rate for the sensors and actuators and C is the coverage. By varying $\lambda_s$ and C we get the following 3D graph.

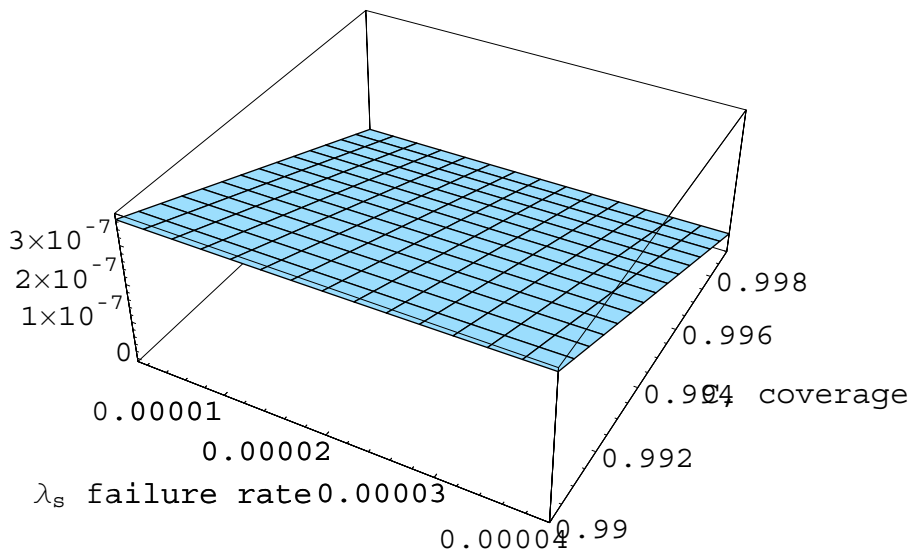


Figure B.4: The probability for losing the critical functionality of the central duplex system.

### B.3.2 Partially-Distributed System

#### Simplex

In Figure B.5 we can the system configuration for the partially-distributed system.

In this system the **Central Node** has the same reliability as in the central system, see equation (B.10).

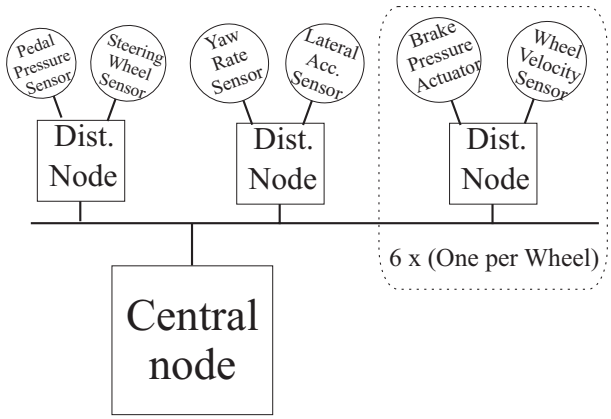


Figure B.5: The partially-distributed system configuration.

The bus will have fewer connections since the sensors and actuators are not connected directly to the node. In the partially-distributed system we assume that additional sensors and actuators are connected to existing distributed nodes or possibly the central node. Thus we will have a total of 9 bus connections, i.e., $n = 9$.

$$\lambda_{HDbus} = \frac{n\,\lambda_{con}}{10}$$

$$R_{HDbus} = e^{-\lambda_{HDbus}} = 0.9999988$$

The distributed nodes, i.e., nodes with sensors and actuators connected are the following. Two nodes with two sensors connected. Which have the following failure rate, the node with yaw rate and lateral velocity sensors and the node with steering wheel sensor and brake pedal has $\lambda_{HDylv}$ and $\lambda_{HDswbp}$ per hour. Which are.

$$\lambda_{HDylv} = \lambda_{HDswbp} = \lambda_{\mu P2} + \lambda_{PIC} + \lambda_{CI} + \lambda_{BD} + \lambda_{con} + 2\,\lambda_s$$

For the six wheel nodes with a wheel velocity sensor and a brake pressure actuator connected $\lambda_{HDwvbp}$ as failure rate.

$$\lambda_{HDwvbp} = \lambda_{\mu P2} + \lambda_{PIC} + \lambda_{CI} + \lambda_{BD} + \lambda_{con} + 2\,\lambda_s$$

The we can get the **System reliability for the total functionality** from these components in a series system.

$$R_{hd} = R_{Cnode}\,R_{HDbus}\,e^{-\lambda_{HDylv}}\,e^{-\lambda_{HDswbp}}\left(e^{-\lambda_{HDwvbp}}\right)^6 = 0.999893 \times e^{-16\,\lambda_s}$$

The probability of loss of functionality per hour is thus:

$$Q_{hd} = 1 - R_{hd}$$

In Figure B.6 we see the probability of loss of functionality when the failure rate for sensors and actuators, i.e., $\lambda_s$ and $\lambda_a$, is ranging $10^{-6}$ to $40 \times 10^{-6}$.

### Critical Functionality

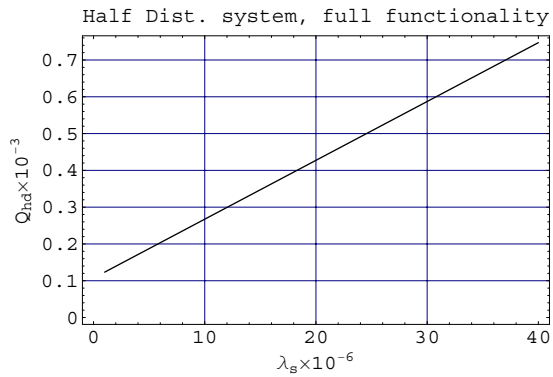


Figure B.6: The probability of losing any functionality in the partially-distributed system

When calculating the probability of loss of **Critical functionality** we only need the brake pedal sensor and one brake pressure actuator on each side of the vehicle, the processing power for only the brake functionality is sufficient in the distributed nodes and will not requre the Central node. For the pedal sensor node, we have a failure rate and a reliability of (HDC = Partially-Distributed Critical System):

$$\lambda_{HDCpedal} = \lambda_{\mu P2} + \lambda_{PIC} + \lambda_{CI} + \lambda_{BD} + \lambda_{con} + \lambda_s$$

$$R_{HDCpedal} = e^{-\lambda_{HDCpedal}}$$

For the wheel nodes with brake presssure actuator:

$$\lambda_{HDCbrake} = \lambda_{\mu P2} + \lambda_{PIC} + \lambda_{CI} + \lambda_{BD} + \lambda_{con} + \lambda_a$$

$$R_{HDCbrake} = e^{-\lambda_{HDCbrake}}$$

We have a parallel system on each side where at least one of the actuator node must be functional for acceptable behavior. Thus, the reliability that at least one brake node at one side is working is $R_{HDCside}$.

$$R_{HDCside} = 1 - (1 - R_{HDCbrake})^3$$

The total reliability for the critical functionality is (serie system with bus, pedal node, 2*brake pressure act.):

$$R_{HDC} = R_{HDbus}\, R_{HDCpedal}\, R_{HDCside}^2$$

The probability for system failure is $Q_{CHD}$:

$$Q_{HDC} = 1 - R_{HDC} = 1 - 0.999986 \times e^{-\lambda_s} \left( \left( 1 - 0.999987 \times e^{-\lambda_s} \right)^3 - 1 \right)^2$$

The probability of losing critical functionality when $\lambda_a$ and $\lambda_s$ is $10^{-6}$ to $40 \times 10^{-6}$.

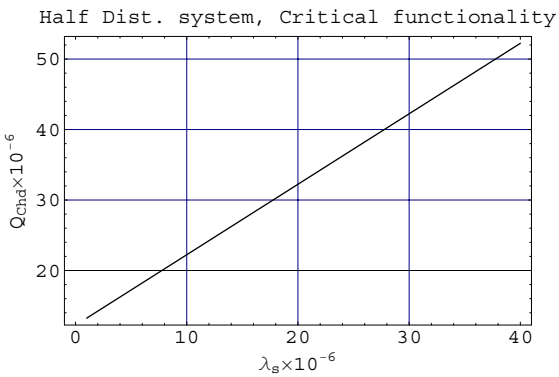


Figure B.7: The probability of losing critical functionality in the partially-distributed system

### Partially-Distributed, Duplex

Only the critical part of the partially-distributed system will be duplicated when we increase the dependability of this system. The system part that will be duplex is the bus, the node with the pedal sensors connected and the brake actuator nodes we will take advantage of the fact that there already is an inherent redundancy for those nodes. For this node we will have these same Markov model as for the central node in

the central system. Thus we have a coverage C for detecting failures in this case also. The probability of failure per hour is for the pedal node:

$$P_F(t) = 1 + (2\,C - 1)\,e^{-2\,t\,\lambda} + 2\,C\,e^{-t\,\lambda} \tag{B.20}$$

The failure rate of the pedal node is:

$$\lambda_{pedalHDD} = \lambda_{\mu P2} + \lambda_{PIC} + \lambda_{CI} + \lambda_{BD} + \lambda_{con}$$

Reliability for pedal node in the Partially-Distributed Duplex (HDD) system $R_{pedalHDD} = 1 - P_F(1)$ with $\lambda$ in equation (B.20) as $\lambda_{pedalHDD}$.

We will again assume that we have three pedal sensors. These sensors are connected to both pedal nodes. They do not fail in a way that affect the other sensors. However, we assume that we need two of three sensors for correct operation. The reliability of a 2-of-3 system is (with independent components with the same reliability R) $3\,R^2 - 2\,R^3$.

$$R_{SensorsHDD} = 3\,e^{-2\,\lambda_s} - 2\,e^{-3\,\lambda_s}$$

For the duplex bus we have:

$$R_{BusHDD} = 1 - (1 - R_{HDbus})^2$$

Finally for the brake pressure actuators we have the same as with the central system:

$$R_{BrakeActHDD} = R^2_{oneside}$$

The reliability for the system is finally a serie system of: the pedal sensors - the pedal node - the bus - the actuators.

$$R_{HDD} = R_{sensorsHDD}\,R_{pedalHDD}\,R_{BusHDD}\,R_{BrakeHDD}$$

The probability for system failure per hour is thus:

$$Q_{HDD} = 1 - R_{HDD} = 1 - \left(0.999975 + 2.51 \times 10^{-5}\right) \left(3\,e^{-2\,\lambda_s} - 2\,e^{-3\,\lambda_s}\right) \left(1 - \left(1 - e^{-2.966\times10^{-6} - \lambda_s}\right)^3\right)^2$$

In Figure B.8 the probability of system failure is shown when varying $\lambda_s$ and C,

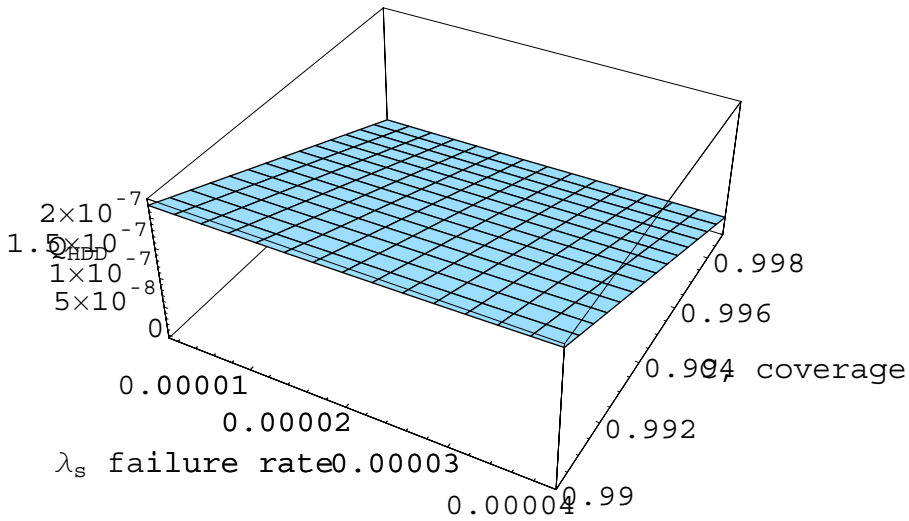


Figure B.8: The probability for losing the critical functionality of the partially-distributed duplex system.

### B.3.3 Distributed System

#### Simplex

In Figure B.9 we show the system configuration for the fully-distributed system. In this system there is no Central Node.

The bus will almost have the same amount of connections as the partially-distributed system, except the Central node. In the distribute system we assume that additional sensors and actuators are connected to existing distributed nodes. Thus we will have a total of 8 bus connections, i.e., $n = 8$.

$$\lambda_{DSbus} = \frac{n\,\lambda_{con}}{10}$$

$$R_{DSbus} = e^{-\lambda_{DSbus}} = 0.9999989$$

The distributed nodes, i.e., nodes with sensors and actuators connected are the following. Two nodes with two sensors connected. They

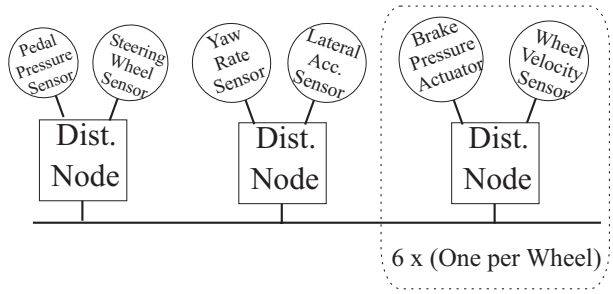


Figure B.9: The fully-distributed system configuration.

have the following failure rate, the node with yaw rate and lateral velocity sensors and the node with steering wheel sensor and brake pedal has $\lambda_{DSylv}$ and $\lambda_{DSswbp}$, respectively, (per hour).

$$\lambda_{DSylv} = \lambda_{DSswbp} = \lambda_{\mu P3} + \lambda_{PIC} + \lambda_{CI} + \lambda_{BD} + \lambda_{con} + 2\,\lambda_s$$

For the six wheel nodes with a wheel velocity sensor and a brake pressure actuator connected $\lambda_{DSwvbp}$ as failure rate.

$$\lambda_{DSwvbp} = \lambda_{\mu P3} + \lambda_{PIC} + \lambda_{CI} + \lambda_{BD} + \lambda_{con} + \lambda_s + \lambda_a$$

The we can get the **System reliability for the total functionality** from these components in a series system.

$$R_{DS} = R_{DSbus}\, e^{-\lambda_{DSylv}}\, e^{-\lambda_{DSswbp}} \left(e^{-\lambda_{DSwvbp}}\right)^6 = 0.999934 \times e^{-16\,\lambda_s}$$

The probability of loss of functionality per hour is thus:

$$Q_{DS} = 1 - R_{DS}$$

And in Figure B.10 we see the probability of loss of functionality when the failure rate for sensors and actuators, i.e., $\lambda_s$ and $\lambda_a$, is ranging $10^{-6}$ to $40 \times 10^{-6}$.

### Critical Functionality

Here we assume that the system only need the brake pedal sensor and one brake pressure actuator on each side of the vehicle. For the pedal sensor node in the Distributed Critical System (DCSpedal) we have:

$$\lambda_{DCSpedal} = \lambda_{\mu P3} + \lambda_{PIC} + \lambda_{CI} + \lambda_{BD} + \lambda_{con} + \lambda_s$$

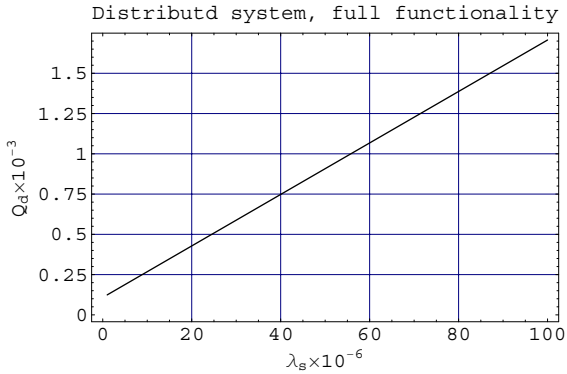


Figure B.10: The probability of losing any functionality in the fully-distributed system

$$R_{DCSpedal} = e^{-\lambda_{DCSpedal}}$$

For the wheel nodes with brake presssure actuator:

$$\lambda_{DCSbrake} = \lambda_{\mu P3} + \lambda_{PIC} + \lambda_{CI} + \lambda_{BD} + \lambda_{con} + \lambda_a$$

$$R_{DCSbrake} = e^{-\lambda_{DCSbrake}}$$

We have a parallel system on each side where at least one of the actuator node must be functional for accepteble behavior. Thus, the reliability that at least one brake node at one side is working is $R_{DCSside}$.

$$R_{DCSside} = 1 - (1 - R_{DCSbrake})^3$$

The total reliability for the critical functionality is (serie system with bus, pedal node, 2*brake pressure act.):

$$R_{DCS} = R_{DCbus}\, R_{DCSpedal}\, R_{DCSside}^2$$

The probability for system failure is $Q_{DCS}$:

$$Q_{DCS} = 1 - R_{DCS} = 1 - 0.999982 \times e^{-\lambda_s} \left( \left( 1 - 0.999984 \times e^{-\lambda_s} \right)^3 - 1 \right)^2$$

The probability of losing critical functionality when $\lambda_a \lambda_s$ is $10^{-6}$ to $40 \times 10^{-6}$ is shown in Figure B.11.

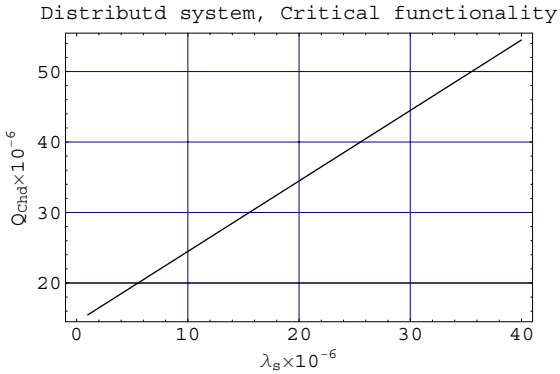


Figure B.11: The probability of losing critical functionality in the partially-distributed system

### Distributed, Duplex

With the duplex partially-distributed system we assume that we only duplicate the critical part of the system. For the duplex system, which will be very similar to the partially-distributed duplex case. Thus, it will contain the bus and the node with the pedal sensors connected and at least one actuator node at each side of the vehicle. For this node we will have the same Markov model as for the central node in the central system. Thus we have a coverage C for detecting failures in this case also.

For one hour we get the probability of failure for the pedal node of:

$$P_F(t) = 1 + (2\,C - 1)\,e^{-2\,t\,\lambda} + 2\,C\,e^{-t\,\lambda} \tag{B.21}$$

The failure rate of the pedal node is:

$$\lambda_{pedalDDS} = \lambda_{\mu P3} + \lambda_{PIC} + \lambda_{CI} + \lambda_{BD} + \lambda_{con}$$

Reliability for pedal node in the fully-Distributed Duplex System (DDS) $R_{pedalDDS} = 1 - P_F(1)$ with $\lambda$ in equation (B.21) as $\lambda_{pedalDDS}$.

We will again assume that we have three pedal sensors. These sensors are connected to both pedal nodes. They do not fail in a way that affect the other sensors. However, we assume that we need two of three sensors for correct operation. The reliability of a 2-of-3 system is (with independent components with the same reliability R) $3\,R^2 - 2\,R^3$.

$$R_{SensorsDDS} = 3\,e^{-2\,\lambda_s} - 2\,e^{-3\,\lambda_s}$$

For the duplex bus we have:

$$R_{BusDDS} = 1 - (1 - R_{DSbus})^2$$

Finally for the brake pressure actuators we have the same as with the central system:

$$R_{BrakeActDDS} = R^2_{oneside}$$

The reliability for the system is finally a serie system of: the pedal sensors - the pedal node - the bus - the actuators.

$$R_{DDS} = R_{sensorsDDS} \times R_{pedalDDS} \times R_{BusDDS} \times R_{BrakeDDS}$$

The probability for system failure per hour is thus:

$$Q_{DDS} = 1 - R_{DDS} =$$

$$1 - \left(0.999967 + 3.29 \times 10^{-5}\right)\left(3\,e^{-2\,\lambda_s} - 2\,e^{-3\,\lambda_s}\right)$$

$$\left(1 - \left(1 - e^{-2.866\times10^{-6} - \lambda_s}\right)^3\right)^2$$

In Figure B.12 the probability of system failure is shown when varying $\lambda_s$ and C.

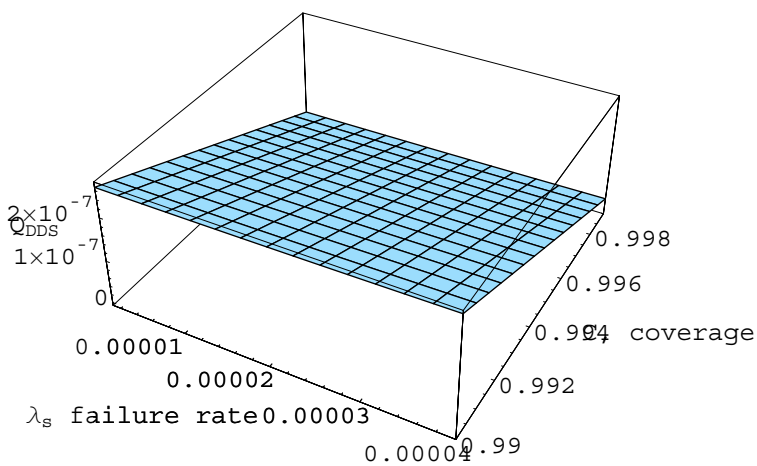


Figure B.12: The probability for losing the critical functionality of the Fully-Distributed duplex system.

# Bibliography


[ARI95] Aeronautical Radio, Inc. *Multi-Transmitter Data Bus, Part 1, Technical Description*, Dec. 1995. 8, 18, 74

[ATJ99] Kristina Ahlström, Jan Torin, and Rikard Johansson. Future electrical flight control systems, analysis of distributed architectures. Technical Report 99-25, Department of Computer Engineering, Chalmers University of Technology, 1999. 109, 133

[BES+01] Josef Berwanger, Christian Ebner, Anton Schedl, Ralf Belschner, Sven Fluhrer, Peter Lohrmann, Emmerich Fuchs, Dietmar Millinger, Michael Sprachmann, Florian Bogenberger, Gary Hay, Andreas Krüger, Mathias Rausch, Wolfgang O. Budde, Peter Fuhrmann, and Robert Mores. FlexRay - the communication system for advanced automotive control systems. In *SAE 2001 World Congress*, SAE TECHNICAL PAPER SERIES, Detroit, Michigan, 2001. 8

[BPG] Josef Berwanger, Martin Peller, and Robert Griessbach. Byteflight - a new high-performance data bus system for safety-related applications. http://www.byteflight.com/. 8

[Byt99] Byteflight specification, ver. 0,5. http://www.byteflight.com/, 29.10.1999 1999. 8

[CAN91] Robert Bosch GmbH. *CAN Specification Version 2.0*, 1991. 10, 17, 52, 85

[CGS00] Vilgot Claesson, Magnus Gäfvert, and Martin Sanfridsson. Proposal for a distributed computer control system in heavy-duty trucks. Dicosmos Internal Report 00-16, Computer Engineering, Chalmers University of Technology., 2000. 15

[Cla99] Vilgot Claesson. *Cost Effective Communication Services for Applications in Distributed Time Triggered Real-Time Sys-*

*tems*. Thesis for the degree of licentiate of engineering, Chalmers University of Technology, 1999. 12

[CPR+92] M. Chrque, D. Powell, P. Reynier, J.-L. Richier, and J. Voiron. Active replication in delta-4. In *IEEE Workshop on Fault-Tolerant Parallel and Distributed Systems*, pages 28 –37, 1992. 5

[CPS98] Vilgot Claesson, Stefan Poledna, and Jan Söderberg. The XBW model for dependable real-time systems. In *International Conference on Parallel and Distributed Systems*, pages 130–138, Tainan, Taiwan, 1998. 12

[DPC+96] P. Dowd, J. Perreault, J. Chu, D.C. Hoffmeister, R. Minnich, D. Burns, F. Hady, Y.J. Chen, M. Dagenais, and D. Stone. LIGHTNING network and systems architecture. *IEEE/OSA Journal Of Lightwave Technology*, 14(6):1371–1387, 1996. 19, 48

[EJ01] Cecilia Ekelin and Jan Jonsson. Evaluation of search heuristics for embedded system scheduling problems. In *Proc. of the International Conference on Principles and Practice of Constraint Programming*, pages 640–654, Paphos, Cyprus, November 16–December 1, 2001. 58

[Fol99] Peter Folkesson. *Assessment and Comparison of Physical Fault Injection Techniques*. PhD thesis, Chalmers University of Technology, 1999. 5, 118

[GSC00] Magnus Gäfvert, Martin Sanfridsson, and Vilgot Claesson. Truck model for yaw and roll dynamics control. Technical Report ISRN LUTFD2/TFRT–7588–SE, Department of Automatic Control, Lund Institute of Technology, Sweden, Sep 2000. 100

[Joh89] Barry W. Johnson. *Design and analysis of fault-tolerant digital systems*. Addison-Wesley series in electrical and computer engineering. Addison-Wesley, cop., 1989. 109, 111

[KDK+89] H. Kopetz, A. Damm, C. Koza, M. Mulazzani, W. Schwabl, C. Senft, and R. Zainlinger. Distributed fault-tolerant real-time systems: The MARS approach. *IEEE Micro*, 9(1):25–40, 1989. 5

[KG94] H. Kopetz and G. Grunsteidl. TTP- a protocol for fault-tolerant real-time systems. *IEEE Computer*, 27(1):14–23, 1994. 19, 21, 22, 31, 45, 72, 78

[KKH+96] H. Kopetz, A. Krüger, R. Hexel, D. Millinger, R. Nossal, R. Pallierer, and C. Temple. Redundancy management in the time-triggered protocol. Technical Report 4/1996, Technical University of Vienna, 1996. 19, 21

[Kop93] H. Kopetz. Should responsive systems be event-triggered or time triggered? *IEICE Trans. on Information and systems*, E76D(11):1325–1332, 1993. 18

[KS80] L. Kleinrock and M. O. Scholl. Packet switching in radio channels: New conflict-free multiple access schemes. *IEEE Transaction Communication*, COM-28(7):1015–1029, 1980. 7, 8

[kSkJC01] Håkan Sivencrona, Lars-Åke Johansson, and Vilgot Claesson. A novel bit-oriented communication concept for distributed real-time systems, qrcontrol. In *3rd International Conference on Control and Diagnostics in Automotive Applications (CDAUTO01)*, Sestri Levante (Genova), Italy., 2001. 76

[KSY84] James F. Kurose, Mischa Schwartz, and Yechiam Yemini. Multiple-access protocols and time-constrained communication. *Cumputing Surveys*, 16(1):43–70, 1984. 5, 7

[KU95] P. J Koopman and B. P. Upender. Time division multiple access without a bus master. Technical Report RR-9500470, United Technologies Research Center, USA, 1995. 19

[LA99] Henrik Lönn and Jakob Axelsson. A comparison of fixed-priority and static cyclic scheduling for distributed automotive control applications. In *1th Euromicro Conference on Real-Time Systems*, pages 142–149, York, 1999. 52

[LH01] G. Leen and D. Heffernan. Time-triggered controller area network. *IEE Computing & Control Engineering Journal*, 12(6):245–256, December 2001. 4

[Lön99a] Henrik Lönn. Initial synchronization of TDMA communication in distributed real-time system. In *19th IEEE International Conference on Distributed Computing Systems (ICDCS'99), Austin, TX, USA*, pages 370–379, 1999. 25, 48

[Lön99b] Henrik Lönn. A simulation model of the DACAPO protocol. Technical Report 99-1, Department of Computer Engineering, Chalmers University, 1999. 138

[Lön99c] Henrik Lönn. *Synchronization and Communication Results in Safety-Critical Real-Time Systems.* Ph.d thesis, Chalmers University of Technology, 1999. 48

[LS95] H. Lönn and R. Snedsbøl. Synchronisation in safety-critical distributed control systems. In *IEEE International Conference on Algorithms and Architectures for Parallel Processing, ICA3PP*, volume 2, pages 891–899, Brisbane, Australia, 1995. 3

[MB76] Robert M. Metcalfe and David R. Boggs. Ethernet: distributed packet switching for local computer networks. *Communications of the ACM*, 19(7):395 – 404, 1976. 5, 10

[MIL92] Military handbook: reliability prediction of electronic equipment, MIL-HDBK-217F, 1992. 109, 111, 113, 143

[MZ95] Nicholas Malcolm and Wei Zhao. Hard real-time communication in multiple-access networks. *Real Time Systems*, 9(1):75–107, 1995. 5

[Pol99] Projektbeskrivning pålbus - utvärdering av pålitliga distribuerade styrsystem. SP Swedish National Testing and Research Institute, Borås, Sweden, Technical notes ("Arbetsrapport") SP-AR 1999:31,, 1999. vii

[Pow92] D. Powell. Failure mode assumptions and assumption coverage. In *Twenty-Second International Symposium on Fault-Tolerant Computing, FTCS-22.*, pages 386–395, 1992. 5

[Pra95] Dhiraj K. Pradhan. *Fault-Tolerant Computer System Design.* Prentice-Hall, Inc., 1995. 109

[PSA97] D. Peng, K. G. Shin, and T. Abdelzaher. Assignment and scheduling of communicating periodic tasks in distributed real-time systems. *IEEE Trans. on Software Engineering*, 23(12):745–758, December 1997. 57

[PT86] K.J. Perry and S. Toueg. Distributed agreement in the presence of processor and communication faults. *IEEE Transactions on Software Engineering*, 12(3):477–482, 1986. 5

[RLST95] Babak Rostamzadeh, Henrik Lönn, Rolf Snedsbøl, and Jan Torin. DACAPO: a distributed computer architecture for safety-critical control applications. In *Intelligent Vehicles Symposium*, pages 376 – 381, Detroit, MI, USA, 1995. 22, 31, 46, 48

[SCG00] Martin Sandfridsson, Vilgot Claesson, and Magnus Gäfvert. Investigation and requirements of a computer control system in a heavy-duty truck. Technical Report TRITA-MMK 2000:5, ISSN 1400-1179, ISRN/MMK–00/5–SE, Mechatronics Lab, Royal Institute of Technology., 2000. 110, 117, 121

[SHW94] Neeraj Suri, Michelle M. Hugue, and Chris J. Walter. Synchronization issues in real-time systems. *Proceedings of the IEEE*, 82(1):41–54, 1994. 3, 93

[SS92] Daniel P. Siewiorek and Robert S. Swarz. *Reliable Computer Systems, design and evaluation.* Digital Press, second edition, 1992. 109

[Sta91] W. Stallings. *Data and Computer Communications.* Macmillan Publishing Company, New York, 1991. 19

[TB94] K. Tindell and A. Burns. Guaranteed message latencies for distributed safety-critical hard real-time control net. Technical Report YCS 229, Department of computer science, real-time systems research group, University of York, 1994. 18

[TBW92] K. W. Tindell, A. Burns, and A. J. Wellings. Allocating hard real-time tasks: An NP-hard problem made easy. *Real-Time Systems*, 4(2):145–165, June 1992. 57

[TBW95] Ken Tindell, Alan Burns, and Andy J. Wellings. Analysis of hard real-time communications. *Real-Time Systems*, 9(2):147–171, 1995. 52

[Tem98] C. Temple. Avoiding the babbling-idiot failure in a time-triggered communication system. In *Twenty-Eighth Annual International Symposium on Fault-Tolerant Computing*, pages 218 – 227, 1998. 5

[TL94] Martin Törngren and Hans Lind. On decentralization of control functions for distributed real-time motion control. In *International Symposium on Robotics and Manufacturing, ISRAM'94*, 1994. 106

[TTP99] Time-Triggered Technology, TTTech Computertechnik GmbH, www.tttech.com. *TTP/C protocol, Specification of the Basic TTP/C protocol*, 1.0 edition, Jul 1999. 31, 45, 46, 72, 78, 93

[X-B98] X-By-Wire Team. X-By-Wire - safety related fault tolerant systems in vehicles, final report, Nov. 1998. vii, 12

[Yeh96] Y.C. Yeh. Triple-triple redundant 777 primary flight computer. In *Aerospace Applications Conference*, volume 1, pages 293 –307, 1996. 8