Schedulability-Driven Reliability Optimization of FlexRay Communications

Xiaojun Ma

Informatics and Mathematical Modelling,
Technical University of Denmark

March 17, 2008

Contents

List of Figures

List of Tables

Acknowledgements

This master thesis brought me many challenges during the 6 months. With helps and contributions of many people, I would like to thank the ones who contributed in various ways.

First and foremost, I would like to thank my supervisor, Paul Pop, for his guidance, concern and help through my master thesis. He proposed the idea of the message scheduling optimization approach and introduced me the modeling language for mathematical programming. He always encouraged and supported me warmly when I was face difficulties. His advice and confidence are very important for me in the completion of this thesis. I am grateful to have the pleasure of knowing and working with him.

Secondly, I own my gratitude to my family and friends. Although my parents lived far from me, they always gave me their love, constant support, concern and encouragement.

Finally, I want to give my special thanks to Di Wang for his trust, patience and support through my master thesis. He gave me many useful suggestions, and he

always encourage me when I was tired and negative.

Thank you for all the people that helped me with this thesis. I will be truly glad to share my accomplishment in the future.

Abstract

Currently, more and more real-time systems are implemented on distributed architectures in order to meet reliability, functional, and performance constraints. Communications in such systems can be triggered either dynamically, in response to an event (event-driven), or statically, at predetermined moments in time (time-driven).

A large consortium of automotive manufacturers and suppliers has recently proposed a protocol called FlexRay, which allows the sharing of the bus among event-driven (ET) and time-driven (TT) messages, thus offering the advantages of both TT and ET worlds. FlexRay will very likely become the de-facto standard for in-vehicle communications. While the importance of FlexRay has been quickly recognized, analysis and optimization approaches for the protocol have not been available until recently.

FlexRay will be used more and more in safety-critical applications, where message transmission has to be reliable. There are several approaches to increase message transmission reliability, such as the hardware replication and re-execution.

The thesis will investigate how to get an optimal message scheduling result in the

static and dynamic segments of FlexRay protocol, and how to increase the message reliability using relevant approaches. The objective of the thesis is to determine a combination of redundancy techniques for each message in an application, such that the message transmission meets the reliability goal imposed by the designer, and the application is schedulable.

Introduction

In the early days of automotive electronics, messages were exchanged through point-to-point links between electronic control units (ECUs). However, this mechanism was unable to deal with an exponential increasing use of ECUs due to the problem of weight, cost, complexity and reliability caused by the wires and connectors [23]. To solve this problem, vehicle networks are considered to be used for multiplex message transmissions, where the rules — communication protocols are needed to ensure well-balanced transmissions and manage the message access to the vehicle network.

At present, there are several communication protocols for vehicle networks based on the event-triggered or time-triggered mechanism. Controller Area Network (CAN) is currently used in many automotive applications. It is an event-triggered protocol developed by the German company Robert Bosch in 1985 for in-vehicle applications. It dynamically schedules the messages based on the occurrence of events [24]. CAN has a good performance in terms of bandwidth utilization. However, it is hard to deal with safety critical issues because it is difficult to guarantee the message deadlines. Thus CAN is not considered deterministic enough to be used in safety critical applications which require high reliability and dependability [25].

Time-triggered Protocol (TTP) has been developed by Technical University of Vienna for more than 25 years. Messages using TTP are statically scheduled based on the progression of time. It has an advantage that it can precisely control the message transmission and reception time. This characteristic makes it suitable for safety critical applications. However in [23], the author mentioned TTP has three drawbacks: one is the inefficiency in term of network utilization and periodic message response time; another is the lack of flexibility; and the other is the difficulty in extendability.

Nowadays, either an event-triggered or a time-triggered mechanism is required for message transmissions in the vehicle network, and in some cases, both of them are required at the same time in complex control systems [24]. Currently, a large consortium of automotive manufactures has proposed a hybrid type of protocol, FlexRay communication protocol [1]. FlexRay allows to transmit both event-triggered and time-triggered messages on the same bus, thus supplying the advantage of both worlds [2].

FlexRay communication happens within the recurring communication cycle. Each cycle consists of a mandatory static (ST) segment and an optional dynamic (DYN) segment. Message transmissions in the ST segments is based on the time-division multiple-access (TDMA) scheme which is adopted by TTP [25]. Message transmissions in the DYN segment which is similar to Byteflight protocol, is based on the flexible TDMA (FTDMA) bus access scheme [2].

FlexRay is considered to have more potential in supporting additional services for safety critical systems due to its flexibility in configurations. However, there are not many researches proposed analysis techniques and optimization approaches for both ST and DYN segments of FlexRay protocol. In [26], the author proposed an approach to static scheduling for the FlexRay system. But in their discussion, they assumed

FlexRay communications only scheduled in the ST segment based on the TDMA scheme. In [25], the author discussed the performance analysis of the Byteflight protocol, which adopted the same bus access scheme FTDMA like communications in the DYN segment of FlexRay. However, their discussion assumed a quasi-TDMA (QTDMA) technique as the Byteflight access mechanism, which means that they did not fully exploit the FTDMA technique. The transmission of minislots in the DYN segment is also ignored in their discussion.

In this thesis, we investigate how to get an optimal scheduling result in both ST and DYN segments of FlexRay based on the discussion in [2], which presents a static cyclic scheduling technique and a worst-case response time analysis for communications in ST and DYN segments, respectively. Moreover, we propose several techniques for increasing the FlexRay message reliability on the ground that each message is schedulable. It is proved that FlexRay is suitable for safety critical applications, such as x-by-wire system, which generally require for ultra-high reliability demand fault tolerance and extensive redundancy on the communication bus.

Our main objective is to determine a combination of redundancy techniques for each FlexRay message in an application, such that the message meets the reliability goal and the application is schedulable.

This thesis is organized in 6 chapters. Chapter 1 gives all the necessary theoretical concepts and methods used in this work. Chapter 2 discusses two optimization approaches for FlexRay communication in ST and DYN segments, respectively, such that each message is schedulable on a FlexRay bus. Chapter 3 introduces several redundancy techniques to increase the message reliability in FlexRay, such that each message can meet its reliability goal. In Chapter 4, we extend the analysis to schedule the message on two FlexRay channels. Several solutions are also proposed to combine the two optimization approaches together for both ST and DYN messages based on

the knowledge in Chapter 2 and 3. Chapter 5 shows the evaluation analysis to determine the efficiency of our solutions. The last chapter presents our conclusions.

Chapter 1

Background

Introduction

The main purpose of this chapter is to provide the necessary background for the concepts and methods presented in this thesis. First, we give a short overview about vehicle communication networks. Then we introduce FlexRay network topology, node and cycle architecture, FlexRay message schedulability and reliability. We also give a short description about AMPL which is the programming language for our implementation. In the last part of this chapter, we briefly describe the system model used in this thesis.

1.1 Vehicle Communication Networks

Vehicle communication networks are used to connect components in automotive applications. Typical vehicle components are Engine Control Modules(ECM), Transmission Control Modules(TCM), Anti-lock Brake Systems(ABC) and so on[13]. Figure 1.1 shows a typical CAN network in a car.

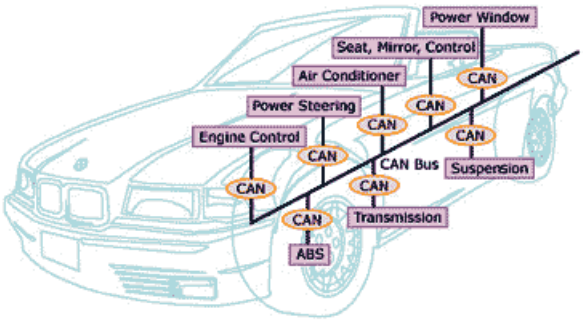


Figure 1.1: A typical CAN network in a car from [13].

Each module or component can be seen as a node in the vehicle network. Communications between these nodes need some rules—standard protocols, to ensure well-balanced message transmission performance. Currently, vehicle network communication is mainly based on the event-triggered (ET) and time-triggered (TT) mechanism. The event-triggered protocol, such as Controller Area Network (CAN), manages message transmission based on the occurrence of events. The message access to the network is dynamically based on the message priorities. The time-triggered protocol (TTP) uses TDMA (Time Division Multiple Access) scheme and timing points to control nodes to access the communication bus. It is also possible to combine these two kinds of mechanisms in one protocol, such as FlexRay, which allows the ET and TT messages to transmit on the same channel.

1.2 FlexRay Protocol

FlexRay is a new protocol which is developed in 1999 by the core members of the FlexRay Consortium (BMW, Bosch, DaimlerChrysler, Freescale, GM, NXP Semiconductors and Volkswagen) for future in-vehicle communication systems, specially for safety-critical applications. The aim of FlexRay is to develop an advanced communication system for high-speed control applications in vehicles such as advanced powertrain, chassis and by-wire systems, to provide flexibility, reliability and functional alternatives [14].

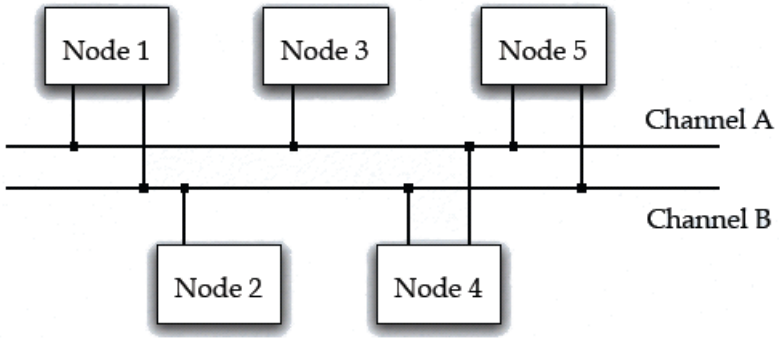
1.2.1 Network Topology

As its name, FlexRay has flexibility and scalability features. It can not only support bus and multiple star network topology in Figure 1.2, but also combine these two topologies for comprehensive systems.

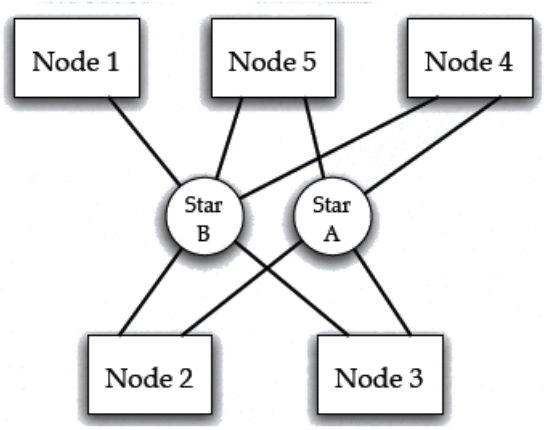
FlexRay supports two channels for data communication and every channel can provide a gross data rate of 10Mbits/s. It increases the net bandwidth 20 times in advanced automotive control applications compared to CAN protocol. The two independent FlexRay channels can also be used as a redundant communication system for increasing error containments.

1.2.2 Node Architecture

Each node connected to a FlexRay bus contains a host, a communication controller (CC) and a bus guardian (BG) as displayed in Figure 1.3. The host is an embedded system processor running software to control the communication process. CC is one



(a) Bus topology



(b) Star topology

Figure 1.2: FlexRay network topology from [1]

of the most important parts in a node. It handles the message transmission and reception and maintains message scheduling based on the clock synchronization[1]. The host and CC are interconnected through a controller host interface (CHI) which is used to handle configurations and message buffers for transmission and reception. BG provides the error detection mechanism and generates interrupts when critical problems occur[15].

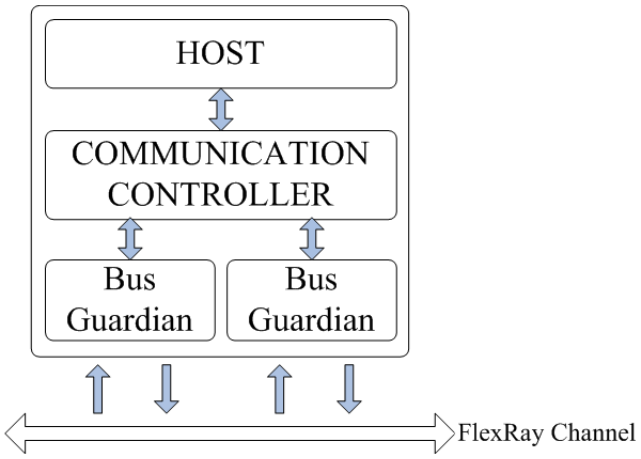


Figure 1.3: FlexRay node architecture.

1.2.3 Communication Cycle Architecture

FlexRay communication happens within the recurring communication cycle. Each cycle consists of four parts shown in Figure 1.4, a mandatory static (ST) segment, a network idle time (NIT), an optional dynamic (DYN) segment and a symbol window [1].

- **Static segment**

  The ST segment is used to transmit ST messages based on the TDMA scheme. A ST segment contains several time slots. Each time slot has the same length.

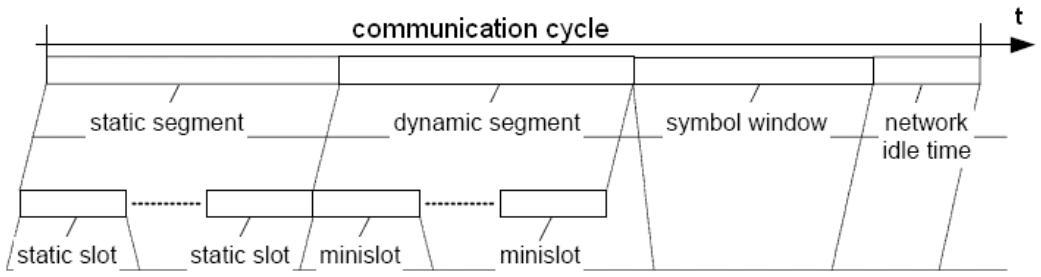


Figure 1.4: FlexRay communication cycle architecture from [1].

The number of slots in a ST segment is deterministic off-line by the designer.

- **Dynamic segment**

  DYN messages are transmitted in the DYN segment. A DYN segment includes a number of "minislots". Each DYN slot consists of several "minislots". Different with the ST slot, the length of each DYN slot is a variable which is depended on the assigned message size.

- **Network idle time**

  NIT is a communication-free period during which the node calculates clock correction terms[1]. In the NIT, nodes in a cluster do not have any message exchanges between them.

- **Symbol window**

  The symbol window is an optional part in the communication cycle. It is used to transmit special messages, called FlexRay symbols.

We are only interested in the ST and DYN segments for message transmissions in this thesis. So we only discuss these two segments in each communication cycle in the following content.

1.2.4 Message Schedulability

When several messages are going to transmit on a FlexRay bus, the designer has to decide a scheduling policy to ensure the message schedulability. The scheduling policy must unambiguously determine the message transmission at any time.

In this thesis, the schedulability analysis of FlexRay messages is based on two different policies for the two types of FlexRay messages, the ST and DYN messages, respectively.

The ST message is seen as a time-triggered (TT) message. The host in each node holds a schedule table with the transmission time for the ST message. When the ST message can access to the FlexRay bus is depended on the starting time point saved in the schedule table. In order to ensure all the ST messages in the application are schedulable, we must make sure each ST message can finish its transmission before its message deadline.

The DYN message is seen as an event-triggered (ET) message. Different from the ST message, the transmissions of DYN messages are not depended on any schedule table. When the DYN message can be transmitted on the FlexRay bus is based on the message priority. In other words, we can not use the time point to control the DYN message transmission as the ST message. It is not easy to ensure that each DYN message is schedulable compared to the ST message. In this thesis, we discuss a heuristic method to get the worst response time of each DYN message based on the timing analysis in [2]. If we can make sure the message worst response time is not more than the message deadline, then the DYN message is schedulable.

1.2.5 Message Reliability

"Reliability is the probability of a device to function correctly over a given period of time under a given set of operating conditions" [19]. A safety critical application must ensure a higher reliability to satisfy fault tolerant requirements. The reliability of a message is defined as the probability that the message performs correctly throughout an interval of time. In our implementation, the time interval of concern is usually depended on the time period from the message transmission starting point to its deadline point.

TTP has several safety services, including the node membership, the clique avoidance algorithm and the independent bus guardian, to guarantee a fail-silent behavior of a faulty node [12]. Compared to TTP, there are not many details about the membership service and message diagnosis in FlexRay specification as [1]. But in [9], the author mentioned each FlexRay node has a local BG for avoiding a faulty node to transmit within unscheduled time slots. There are two watchdogs in BG used to detect illegal behaviors of the synchronization signals [15]. If the CC tries to transmit a message within a wrong time slot or it detects a clock or timing error, the local BG can pass the related illegal information to the node. FlexRay also provides a mechanism to check if the BG works correctly or not.

Unlike the BG in TTP which is independent to the CC, the BG in FlexRay shares the power supply with the CC and it is controlled by the CC. So strictly speaking, it is not totally independent on the CC. From the safety point of view, the dependent mechanism is a drawback for safety requirements [6].

But due to the flexible structure of FlexRay protocol, we discuss several relevant techniques, such as the hardware replication and re-execution, to increase FlexRay message reliability in Chapter 3.

1.3 AMPL and CPLEX

The term "mathematical programming" describes the minimization or maximization of an objective function of numbers of variables and constraints [16]. AMPL, a modeling language for mathematical programming, has been chosen as the programming language for our implementations in this thesis. It allows users to switch several solvers to find an optimal solution.

CPLEX is an optimization solver developed by ILOG software company in 1997. It can solve integer programming and linear programming problems. In this thesis, we choose ILOG CPLEX 10.0.0 optimization solver as an effective solver for our implementation.

To get an optimal solution for our implementations, the following steps are needed[16]:

- Formulate a model, which means several variables, objectives and constraints should be described for the problem.
- Collect the data defined for the specific problem.
- Generate an objective function and several constraint equations from the model and data.
- Solve the problem and find optimal values of the variables.
- Analyze the results.

1.4 System Model

Our system architecture contains several nodes connected to two FlexRay communication channels as shown in Figure 1.5. Each node uses two different message scheduling policies for ST and DYN messages, respectively.

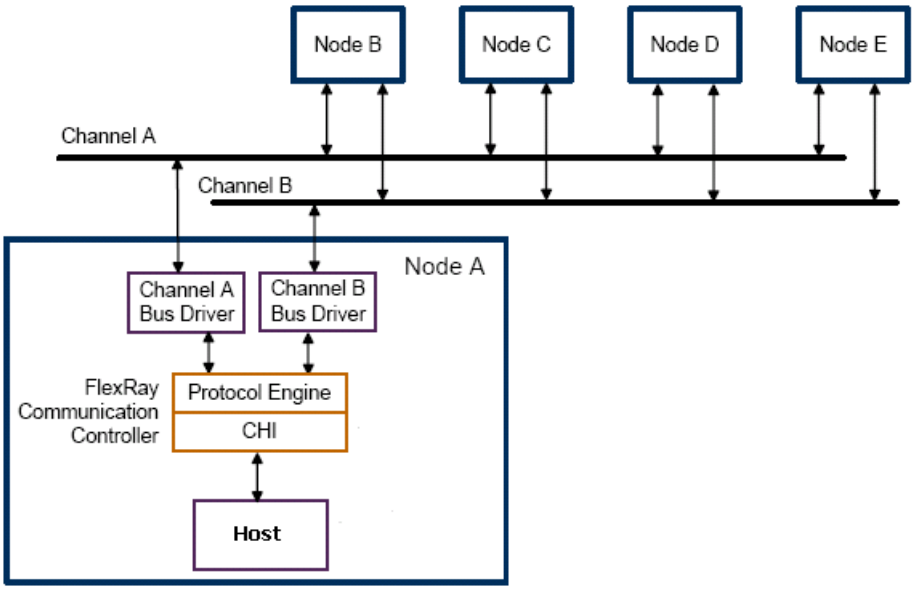


Figure 1.5: FlexRay system architecture from [20].

When several messages are ready to transmit in the CHI on a node, they are assigned to the FlexRay channels through the bus driver. The message assignment is decided by the specified scheduling policy depended on the message attribute (It is a ST or DYN message). The starting time of each ST message is off-line fixed in the schedule table which is saved in the host of the node. The assignment of the DYN message is based on the message priority. The ST message transmission is time-triggered depended on a local clock in each node [3]. The synchronization of local clocks is decided by FlexRay communication protocol [1].

Chapter 2

Message Schedulability Optimization

Introduction

In this chapter, we explain how to guarantee that each message is schedulable in both ST and DYN segments on a FlexRay bus. There are two parts: in the first part, we present how static (ST) messages transmit on a FlexRay communication bus and how to model ST messages scheduling in the ST segment; in the second part, how dynamic (DYN) messages transmit on a FlexRay communication bus will be explained, and the DYN message scheduling based on the schedulability analysis in [2] is modeled as well.

2.1 Static Message Schedulability Optimization

The process of ST message scheduling is used to decide the schedule table for all the ST messages. We need to make sure that each ST message can meet its deadline, then all the ST messages are schedulable.

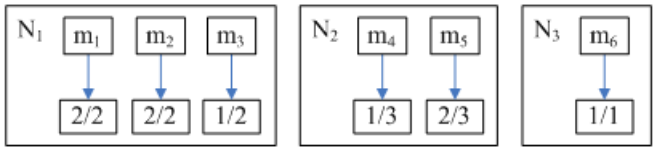
The ST messages are transmitted in the ST segments. The bus for the ST segment is implemented with time division multiple access (TDMA) scheme. The system architecture for the ST segment described in this section is consistent with the ST segment structure of FlexRay protocol.

The goal of this section is to find an optimization approach for the ST message scheduling, such that the total response time for all the ST message is minimized.

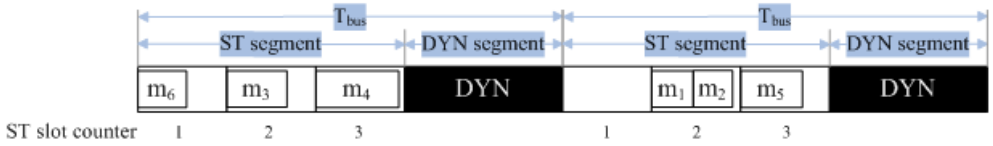
2.1.1 Static Message Transmission

The ST segment has a specified size and a fixed number of time slots. Each time slot has the same length. A ST message is defined as data exchange between different nodes. Each node in the system will be assigned to one or more pre-determined time slots for message transmission in every communication cycle. For simplicity, we assume that each node has only one time slot for transmission in each cycle, which means that the number of time slots in a ST segment is equal to the number of nodes which have ST messages to send. This ensures that each node has at least one chance for transmission in every communication cycle. During the transmission process, the assignment can not be changed.

In Figure 2.1(a), there are three nodes, N_1, N_2 and N_3, sending the ST messages m_1, m_2...m_6 using a FlexRay bus. Each ST message uses a schedule table label,



(a) The schedule table for the ST message



(b) Message assignment for the ST messages.

Figure 2.1: An example for the ST message transmission.

specifying that which time slot and cycle the message should be assigned. For example, m_3 points to a label "1/2", describing that it must be transmitted in the second slot of the first cycle. We can notice that the messages coming from the same node should be transmitted in the same slot of each cycle and the messages coming from the different nodes must be transmitted in the different slots. For example, in Figure 2.1(b), m_1, m_2 and m_3 belonging to N_1 are always sent in the second slot, although the cycle number of these messages are different. m_4 and m_6 are transmitted in the third and first slot of the first cycle. They are always transmitted in the different slots because they are coming from the different nodes.

2.1.2 Static Message Modeling Description

In this part, the model for ST messages scheduling using mathematical programming framework will be given based on the discussions in 2.1.1.

2.1.2.1 Notations

Several notations will be explained in this part, including sets, parameters and variables used in the model.

- **Sets**

STmessage	a set of ST messages
	STmessage = $\{m_i$|i = 1, . . . ,m$\}$
NODE	a set of nodes
	NODE = $\{n_i$|i = 1, . . . ,n$\}$
NM	a set of node-message pairs describing the affiliations between nodes and messages
	$\{(n_i, m_j)\|n_i \in$ NODE, $m_j \in$ STmessage$\}$
CYCLE	a set of cycle numbers
	CYCLE = 1, 2, . . . , c
SLOT	a set of time slot numbers
	SLOT = 1, 2, . . . , s
CS	a set of cycle-slot pairs
	{(c, s)|c ∈ CYCLE, s ∈ SLOT}

- **Parameters**

  There are two kinds of parameters, the ST segment parameters and the ST message parameters, in Table 2.1 and 2.2, respectively.

  A ST segment has a fixed size $STsize$ and several time slots. Each slot has a specified size $Sslot$. $DYNsize$ and $Tbus$ are two parameters described the length of the DYN segment and the whole cycle, respectively.

  A message with deadline $Mdeadline_m$ has a given size $Msize_m$.

ST Segment Parameters	
$Sslot$	the size of the time slot in the ST segment
$STsize$	the size of the ST segment
$DYNsize$	the size of the DYN segment
$Tbus$	the size of the whole communication cycle, which is equal to the sum size of a ST and DYN segment

Table 2.1: The parameters for the ST segment

ST Message Parameters	
$Msize_m$	the size of the ST message, $m \in STmessage$
$Mdeadline_m$	the deadline of the ST message, $m \in STmessage$

Table 2.2: The parameters for the ST message

- **Variables**

  Two binary decision variables are used to describe the assignments between the slots and the messages, the assignments between the slots and the nodes, respectively. The binary variable $message\_assign_{(c,s),(n,m)}$ is used to describe if the node-message pair (n, m) belonging to the set NM is assigned to the cycle-slot pair (c, s). The variable $node\_assign_{s,(n,m)}$ specifies if the slot number s is assigned to the node n .

$$message\_assign_{(c,s),(n,m)} = \begin{cases} 1, & \text{if node-message pair } (n,m) \text{ is assigned} \\ & \text{to the cycle-slot pair } (c,s) \\ 0, & \text{otherwise} \end{cases} \tag{2.1}$$

$$node\_assign_{s,(n,m)} = \begin{cases} 1, & \text{if the slot } s \text{ is assigned to the node } n \\ 0, & \text{otherwise} \end{cases} \tag{2.2}$$

Static Message Variables	
$Rtime$	the response time of the ST message
$Mcycle$	the cycle number of the ST message
$Mslot$	the slot number of the ST message

Table 2.3: The variables for the ST message

The rest three variables are used to present the message response time, the message cycle number and the message slot number, respectively, as Table 2.3.

2.1.2.2 Constraints

The ST message scheduling is feasible if it satisfies all the constraints imposed by the ST segment requirement of FlexRay protocol in [1]:

- Only one message can be transmitted on a FlexRay bus at a time. Collisions of messages lead to an infeasible result.
- Each message can only be assigned to one time slot.
- Messages belonging to the same node should be assigned in the same time slot of each cycle.
- Messages belonging to the different node should be assigned in the different time slot of each cycle.
- The sum of the messages sizes in each time slot cannot exceed the size of the time slot.
- All the messages must meet their deadlines.

In order to build a mathematical model, the above constraints are formulated in the following content.

Message Mutual Exclusion Constraint: Equation 2.3 describes each ST message can be assigned to the only one cycle-slot pair. Equation 2.4 presents each node can be assigned to the only one time slots in each cycle. These two equations together ensure each message should be transmitted once on a FlexRay communication bus.

$$\sum_{(c,s)\in CS} message\_assign_{(c,s),(n,m)} = 1, \qquad (n,m) \in NM \tag{2.3}$$

$$\sum_{s\in SLOT} node\_assign_{s,(n,m)} = 1, \qquad (n,m) \in NM \tag{2.4}$$

Message, Node and Slot Constraints: This part presents four constraints for messages, nodes and slots in the ST segment. Equation 2.5 and Equation 2.6 together ensure that in each cycle, the messages coming from the same node must be assigned to the same slot, and messages coming from the different nodes must be assigned to the different slots. Equation 2.7 ensures that the total sizes of the messages which are assigned in a time slot, cannot exceed the size of the slot. For example, the total sizes of m_1 and m_2 are not larger than the size of the time slot in Figure 2.1(b), so both of them can be assigned in the second slot of the second cycle. Equation 2.8 describes the relationship between the two binary decision variables $message\_assign_{c,s,n,m}$ and $node\_assign_{s,n,m}$. It also ensures that each node is always assigned to a fixed slot number in every cycle.

$$node\_assign_{s,n,m_i} = node\_assign_{s,n,m_j},$$
$$s \in SLOT, (n, m_i) \in NM, (n, m_j) \in NM \tag{2.5}$$

$$\sum_{(n,m)\in NM} node\_assign_{s,n,m} >= 1, \qquad s \in SLOT \tag{2.6}$$

$$\sum_{(n,m)\in NM} message\_assign_{c,s,n,m} * Msize_m <= Sslot, \qquad (c, s) \in CS \tag{2.7}$$

$$\sum_{\substack{(c,s)\in CS,\\(n,m)\in NM}} message\_assign_{c,s,n,m} = \sum_{(n,m)\in NM} node\_assign_{s,n,m},$$
$$s \in SLOT, (n, m) \in NM \tag{2.8}$$

Message Variables Constraints: This part describes three constraints about three ST message variables, respectively, shown in Table 2.3. Equation 2.9 is used to calculate the message cycle number. Equation 2.10 describes the message slot number. The above two equations describes the schedule table value for the ST message. In Equation 2.11, the expression is to calculate the response time of the ST message, describing how much time is needed for the message transmission.

$$Mcycle_m = \sum_{(c,s)\in CS} message\_assign_{c,s,n,m} * c, \qquad (n, m) \in NM \tag{2.9}$$

$$Mslot_m = \sum_{(c,s)\in CS} message\_assign_{c,s,n,m} * s, \qquad (n,m) \in NM \tag{2.10}$$

$$Rtime_m = (Mcycle_m - 1) * Tbus + Mslot_m * Sslot, \qquad m \in STmessage \tag{2.11}$$

Message Deadline Constraints: All the ST messages have to finish their transmission before their deadlines as Equation 2.12.

$$Rtime_m <= Mdeadline, \qquad m \in STmessage \tag{2.12}$$

2.1.2.3 Objective

The objective is represented as Equation 2.13, which is to minimize the total response time for all the ST messages in the system.

$$Minimize \sum_{m\in STmessage} Rtime_m \tag{2.13}$$

2.1.3 Case Study

In this section, we apply our model into a specified case. The case study system contains three nodes with 9 ST messages, which are transmitting on a FlexRay communication bus as Figure 2.2. The data for the message sizes and the deadlines are presented in Table 2.4.

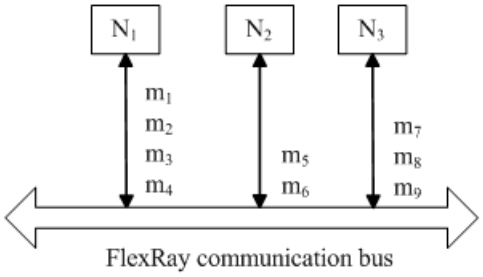


Figure 2.2: The architecture for the case study of the ST message scheduling optimization

Node	Message	Size	Deadline
n_1	m_1	5	40
n_1	m_2	2	60
n_1	m_3	4	70
n_1	m_4	3	90
n_2	m_5	4	20
n_2	m_6	2	50
n_3	m_7	1	50
n_3	m_8	4	70
n_3	m_9	4	90

Table 2.4: The data used for the case study of the ST message scheduling optimization

In this case study, all the messages have fixed sizes and deadlines as Table 2.4. For the safety-critical system, all the ST messages must meet their deadlines. In order to find a scheduling optimization result which has the minimum total response time of all the ST messages, we adjust the size of the time slot until it leads to an infeasible result.

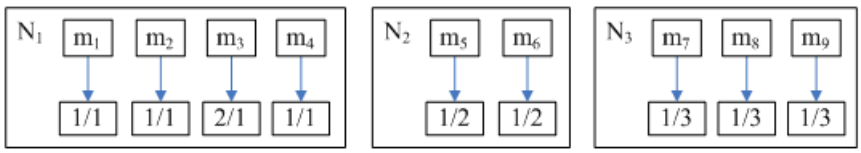
From Equation 2.7, the minimum size of the time slot should be at least equal to the maximum message size (Splitting message for multi-transmissions is not allowed in FlexRay protocol). So at the beginning, we choose 5 (the maximum message size in this case as Table 2.4) as the first value of the parameter $Sslot$. Then every time we increase the parameter value by 1 for the rest cases. We can get the different node sequences for the different cases, and the objective values are also different as shown in Table 2.5. (The sequence of the nodes corresponds to the sequence of the slots they assigned in each cycle. For example, the node sequence "$n_1 n_3 n_2$" describes that in each cycle, the first slot is always assigned to the node n_1, and the second and third slot are assigned to n_3 and n_2, respectively.)

From Table 2.5, we can get a minimum objective value 215 (the total response time for all the 9 ST messages) when $Sslot$ is 10. Under this case, how the ST messages are assigned is displayed in Figure 2.3. When the $Sslot$ value is increased to 17, we get an infeasible result because m_7 fails to meet its deadline in this case.

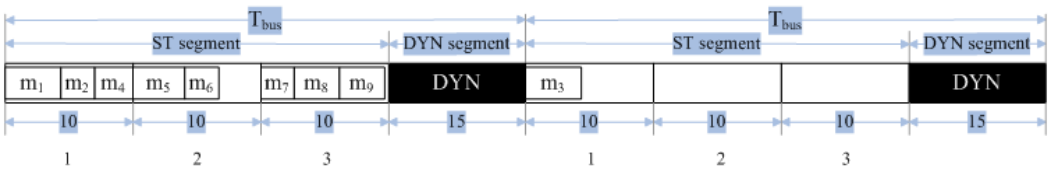
The objective value 215 obtained from Table 2.5 may not be the minimum result. If we give the big enough deadlines for all the ST message, then we can get an even smaller objective value for the total messages response time. In this case, we change all the messages deadlines to 1000 and keep other values the same as Table 2.6. In order to find a minimum result for the total message response time, we choose the value of the slot size in a variable range. In this case, the minimum slot size is 5 as discussed above, and the maximum slot size is 14, which is equal to the maximum

Sslot	Node sequence	Objective value
5	$n_1 n_3 n_2$	230
6	$n_1 n_3 n_2$	228
7	$n_1 n_2 n_3$	227
8	$n_1 n_2 n_3$	253
9	$n_1 n_2 n_3$	237
10	$n_1 n_2 n_3$	215
11	$n_2 n_1 n_3$	257
12	$n_2 n_1 n_3$	279
13	$n_2 n_1 n_3$	301
14	$n_2 n_1 n_3$	266
15	$n_2 n_1 n_3$	285
16	$n_2 n_1 n_3$	304
17	infeasible	infeasible

Table 2.5: The results for the case study of the ST message scheduling optimization



(a) The message schedule table when $Sslot = 10$ in the case study



(b) The message assignment depended on the schedule table

Figure 2.3: The optimal assignment for the case study of the ST message scheduling optimization

sum of a node's message sizes (N_1 has the maximum messages sizes 14 in this case).

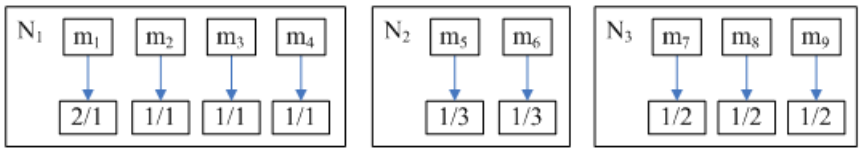
Node	Message	Size	Deadline
n_1	m_1	5	1000
n_1	m_2	2	1000
n_1	m_3	4	1000
n_1	m_4	3	1000
n_2	m_5	4	1000
n_2	m_6	2	1000
n_3	m_7	1	1000
n_3	m_8	4	1000
n_3	m_9	4	1000

Table 2.6: The data used for the case study of the ST messages with big enough deadlines

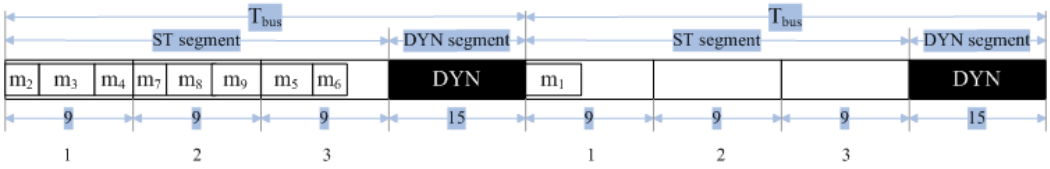
In Table 2.7, we get the results when the $Sslot$ value is between 5 and 14. It is clear that when the size of the slot is equal to 9, we can get a smaller total message response time 186 compared to 215(the minimum result presented in Table 2.5). Under this case, how the messages are assigned is displayed in Figure 2.4.

Sslot	Node sequence	Objective value
5	$n_1 n_3 n_2$	230
6	$n_1 n_3 n_2$	228
7	$n_1 n_3 n_2$	220
8	$n_1 n_3 n_2$	245
9	$n_1 n_3 n_2$	186
10	$n_1 n_3 n_2$	205
11	$n_1 n_3 n_2$	224
12	$n_1 n_3 n_2$	243
13	$n_1 n_3 n_2$	262
14	$n_1 n_3 n_2$	224

Table 2.7: The results for the case study of the ST messages with big enough deadlines



(a) New message schedule tables when $Sslot = 9$ and the deadlines for all the ST messages are 1000



(b) The message assignment depended on the schedule table

Figure 2.4: The optimal message assignment for the case study of the ST messages with the big enough deadlines

2.2 Dynamic Message Schedulability Optimization

The process of DYN message scheduling is to determine a frame identifer for each DYN message and to calculate the message worst response time. Different with the ST messages, we do not know the starting transmission time of the DYN message through the schedule table, which means we are not sure when the DYN message can be transmitted on a FlexRay bus. That is why we call it "dynamic message".

When the DYN message can be transmitted on a FlexRay bus is based on several factors as Equation 2.14 from [2].

$$R_m(t) = \sigma_m + \omega_m(t) + C_m \tag{2.14}$$

Where R_m is the worst response time for the DYN message m. C_m is the message m's transmission time. σ_m is the delay caused by the first factors as discussed below. $\omega_m(t)$ is the worst case delay caused by the second and the third factor.

The first factor indicates that if the DYN message is ready to transmit just after its slot has passed, then it must wait until the next bus cycle for the transmission. The second factor is that the transmission for the DYN message could be delayed by the transmissions for those messages with higher priorities. If several messages are ready in the CHI buffer, which message can be transmitted first is decided by the message priority. The message with a higher priority is allowed to be transmitted first on a FlexRay bus. The last factor means that the message transmission can be delayed by a numbers of "minislots", which is explained in 2.2.1.

In this section, we need to ensure that all the DYN message deadlines are chosen properly based on the worst response time of the DYN messages. In other words, if

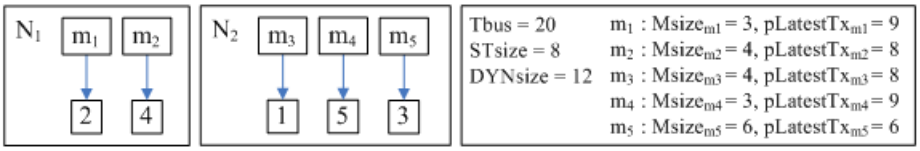
we can make sure that all the DYN messages deadlines are larger than their worst response time, then the DYN messages deadlines are guaranteed, and all the DYN messages are schedulable.

The goal of this section is to find an optimization approach for the DYN message scheduling, such that the total worst response time of all the DYN messages is minimized.

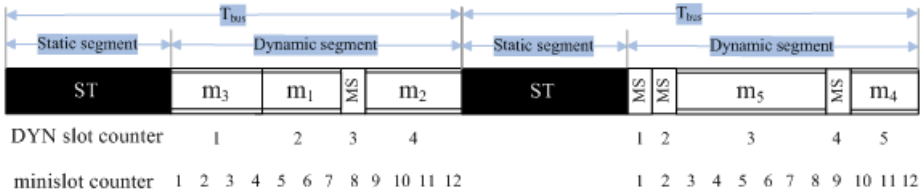
2.2.1 Dynamic Message Transmission

In this section, we will describe how the DYN messages are transmitted on a FlexRay communication bus. Compared to the ST message transmission using TDMA scheme, the flexible time division multiple access (FTDMA) scheme is applied in the DYN segment.

A DYN segment contains a number of "minislots", and each minislot has the same length which is equal to *gdMinislot*[1]. If no messages are transmitted in a DYN slot, then all the nodes should wait for a period of time which is equal to the length of a minislot *gdMinislot*. At the same time, the minislot counter will be incremented by 1. Afterwards, all nodes will check if the current slot counter is equal to their frame identifers. The scheduled message will be transmitted if they are equal. In this case, the length of the slot will be equal to the number of minislots required for the whole message transmission. After that the minislot counter will be incremented again. The above process will be repeated until the end of the DYN segment. If there are no messages or only very few messages to be transmitted within a DYN segment, then there will be a lot of minislots left. Otherwise there will be very few minislots left.



(a) The message assignment for the DYN messages and frame identifers



(b) The DYN messages schedule process

Figure 2.5: An example for the DYN messages transmission

In Figure 2.5(a), there are two nodes N_1 and N_2, which are sending DYN messages m_1 ... m_5 using a FlexRay bus. Node N_1 is assigned to DYN slot 2 and 4, N_2 is given to DYN slot 1, 3 and 5. We can see that a message label always points to a frame identifer number, for example, m_1 points to the frame identifer 2, specifying m_1 needs to be transmitted in DYN slot 2. For DYN messages, a priority scheme[3] is applied to decide which message will be transmitted first if several DYN messages have the same frame identifer. But in this thesis, we assume that each message has its own unique frame identifer, and each frame identifer can be only assigned to one DYN message as Figure 2.5(a). The message with a lower frame identifier has a higher priority for the transmission.

At the beginning of the each DYN cycle, the communication controller for each node will reset the slot and minislot counters. At the beginning of the each slot, the controller will check if there are messages ready for transmission in the controller host interface(CHI) buffer[2], then assign the ready message to frame identifers. In Figure 2.5, we assume all the DYN messages are ready for the transmission before

the first DYN cycle.

In Figure 2.5(b), messages m_3 and m_1 are transmitted in the first and second slot of the first DYN cycle, respectively, according to their frame identifiers. m_5 is supposed to be transmitted in the third slot, but actually there is a minislot replacing its transmission. Because the current minislot counter number 7 is larger than $pLatestTx_{m_5}$ which is equal to 6, m_5 has to wait for the transmission until the next DYN cycle. $pLatestTx_m$ is the last minislot counter number which allows the start of message m's transmission[2]. After passing the minislot in the third slot of the first DYN cycle, the slot counter number is increased to 4 (which is equal to the m_2's frame identifer) and $pLatestTx_{m_2}$ is just equal to the current minislot counter number 8, so m_2 can be transmitted in the fourth slot of the first DYN cycle, although m_5 has a higher priority (lower frame identifer).

2.2.2 Dynamic Message Modeling Description

In this part, the system model for the DYN messages description using mathematical programming framework will be given based on the discussion in 2.2.1.

2.2.2.1 Notations

Several notations will be explained in this part, including sets, parameters and variables used in the system model.

- **Sets**

DYN Segment Parameters	
$STsize$	the size of the ST segment
$DYNsize$	the size of the DYN segment
$Tbus$	the size of one communication cycle
$gdMinislot$	the size of a minislot

Table 2.8: The parameters for the DYN segment

DYNmessage	a set of DYN messages
	DYNmessage = $\{m_i$|i = 1, . . . ,m$\}$
FrameID	a set of frame identifers
	FrameID = 1, 2, . . . , fid

- **Parameters**

  There are two kinds of parameters: the DYN segment parameters and the DYN message parameters in Table 2.8 and Table 2.9, respectively.

  A bus cycle has a deterministic length T_{bus}. In each cycle, it contains one ST segment with a specified size $STsize$ and one DYN segment with a specified size $DYNsize$. Compared to the structure of the ST segment, the DYN segment consists of several minislots, and each minislot has a given size $gdMinislot$.

  A DYN message m with deadline $Mdeadline_m$ has a given size $Msize_m$. $pLatestTx_m$ is a parameter which describes the last minislot number allowing to start message m's transmission in a DYN segment. If the minislot counter number exceeds the $pLatestTx_m$ value, then the current message m must wait to transmit until the next bus cycle.

  $bignum$ is a parameter which is used to describe the relationship between two messages' frame identifiers.

DYN Message Parameters	
$Msize_m$	the size of the DYN message,$m \in DYNmessage$
$Mdeadline_m$	the deadline of the DYN message,$m \in DYNmessage$
$pLatestTx_m$	the last minislot which allows to start the message transmission, $m \in DYNmessage$

Table 2.9: The parameters for the DYN message

- **Variables**

  The binary decision variable $message\_to\_frameID_{fid,m}$ is used to describe whether the DYN message m is assigned to the frame identifer fid or not, $m \in DYNmessage, fid \in FrameID$:

$$message\_to\_frameID_{fid,m} = \begin{cases} 1, & \text{if } m \text{ is assigned to the frame identifer } fid \\ 0, & \text{otherwise} \end{cases} \tag{2.15}$$

  Two binary decision variables $MM_{i,j}$ and $TT_{i,j}$ are used to explain the relationship between any two messages' frame identifers, $i, j \in DYNmessage$:

$$MM_{i,j} = \begin{cases} 1, & \text{if } i\text{'s frame identifer is higher than } j\text{'s} \\ 0, & \text{otherwise} \end{cases} \tag{2.16}$$

$$TT_{i,j} = \begin{cases} 1, & \text{if } i\text{'s frame identifer is lower than } j\text{'s} \\ 0, & \text{otherwise} \end{cases} \tag{2.17}$$

  The rest five variables are used to describe the worst response time of the DYN message, which are displayed in Table 2.10, $m \in DYNmessage$:

DYN Message Response Time Variables	
$Atime_m$	the longest delay in a bus cycle if the message is ready to transmit right after its slot has passed. It has the same meaning with the variable σ_m in [2]
$Btime_m$	the worst delay because of the transmission in the ST segment and those DYN messages with the lower frame identifers. It is has the same meaning with the variable $\omega_m(t)$ in [2]
$Rtime_m$	the worst response time of the DYN message
$BusCycle_m$	the number of bus cycles for message m which should wait because of the transmission of those messages with lower frame identifiers
$roundBusCycle_m$	ceiling of variable $BusCycle_m$

Table 2.10: The variables for the DYN message response time

2.2.2.2 Constraints

The DYN message scheduling is feasible if it satisfies all of the constraints imposed by the DYN segment requirements of FlexRay protocol and the schedulability analysis of DYN messages in [2].

- Only one DYN message can be transmitted on a FlexRay bus at a time. Collisions of message transmissions lead to infeasible results.
- Each DYN message can only be assigned to one frame identifer.
- Each frame identifier can only be assigned to an unique DYN message.
- The DYN message with a lower frame identifer has a higher priority for the transmission.
- All the DYN messages must meet their deadlines.

In order to build a mathematical model, the above constraints are formulated in the following content.

Frame Identifers Constraint: This part presents five constraints about DYN messages' frame identifers.

The following constraint describes that each frame identifer can only be assigned to an unique DYN message in Equation 2.18.

$$\sum_{m \in DYNmessage} message\_to\_frameID_{fid,m} = 1, \qquad fid \in FrameID \tag{2.18}$$

Equation 2.19 calculates the frame identifier value for the DYN message.

$$Mframeid_m = \sum_{fid \in FrameID} message\_to\_frameID_{fid,m} * fid,$$
$$m \in DYNmessage \tag{2.19}$$

Equations 2.20, 2.21 and 2.22 together ensure that there is a deterministic relationship between any two DYN messages' frame identifiers. The parameter $bignum$ is a big enough integer compared to the difference between any two messages' frame identifers in Equation 2.20 and 2.21. For example, in Figure 2.5, m_3 and m_1 have 1 and 2 as their frame identifier, respectively. Thus there is a deterministic relationship between m_3 and m_1, describing m_3 has a lower frame identifier than m_1, and m_1 has a higher frame identifier than m_3. The relationship between any two DYN messages in Figure 2.5 can be described by two binary variables $MM_{i,j}$ and $TT_{i,j}$ in Table 2.11 and 2.12.

$$Mframeid_i - Mframeid_j <= bignum * MM_{i,j},$$
$$i, j \in DYNmessage, i \neq j \tag{2.20}$$

$$Mframeid_j - Mframeid_i <= bignum * TT_{i,j},$$
$$i, j \in DYNmessage, i \neq j \tag{2.21}$$

$$MM_{i,j} + TT_{i,j} = 1, \qquad i, j \in DYNmessage, i \neq j \tag{2.22}$$

Message Worst Response Time Constraints: This part describes five constraints about the DYN message worst response time.

	m_1	m_2	m_3	m_4	m_5
m_1	⁄	0	1	0	0
m_2	1	⁄	1	0	1
m_3	0	0	⁄	0	0
m_4	1	1	1	⁄	1
m_5	1	0	1	0	⁄

Table 2.11: The binary decision variables $MM_{i,j}$ table for Figure 2.5.

	m_1	m_2	m_3	m_4	m_5
m_1	⁄	1	0	1	1
m_2	0	⁄	0	1	0
m_3	1	1	⁄	1	1
m_4	0	0	0	⁄	0
m_5	0	1	0	1	⁄

Table 2.12: The binary decision variables $TT_{i,j}$ table for Figure 2.5.

In Equation 2.27, there is an expression to calculate the worst response time $Rtime_m$ of the DYN message, where $Atime_m$ is the longest delay in one bus cycle if the message m is ready to transmit right after its slot has passed. $Atime_m$ in Equation 2.25 has the same meaning with the variable σ_m in [2].

$Btime_m$ is the worst delay contributed by the transmission in the ST segment and those DYN messages with the lower frame identifiers. $Btime_m$ in Equation 2.26 is supposed to have the same meaning with the variable $\omega_m(t)$ in [2]. But actually they have a small difference. The different place is the value of the variable $BusCycle_m$. In [2], it used the bin covering algorithm to get the $BusCycle_m$ value and modeled the problem of computing $BusCycle_m$ as an integer linear program(ILP). The problem will be very complicated if we use the same way to get the $BusCycle_m$ value in our model. For simplicity, we used Equation 2.23 to get the value of $BusCycle_m$. Here, we assume that those messages with lower frame identifiers than the message m are transmitted before m's transmission in the worst case. So from Equation 2.23, it is known that message m has to wait at least $BusCycle_m$ cycle(s) before its transmission. In Equation 2.24, $roundBusCycle_m$ is the ceiling of the variable $BusCycle_m$.

$$BusCycle_i = \sum_{j \in DYNmessage, i \neq j} MM_{i,j} * Msize_j / pLatesTx_i, \quad i \in DYNmessage \tag{2.23}$$

$$roundBusCycle_m >= BusCycle_m, \qquad m \in DYNmessage \tag{2.24}$$

$$Atime_m = Tbus - STsize - (Mframeid_m - 1) * gdMinislot,$$
$$m \in DYNmessage \tag{2.25}$$

$$\begin{aligned} Btime_m &= roundBusCycle_m * Tbus + STsize + pLatestTx_m \\ &\quad *gdMinislot, \qquad m \in DYNmessage \end{aligned} \tag{2.26}$$

$$Rtime_m = Atime_m + Btime_m + Msize_m, \qquad m \in DYNmessage \tag{2.27}$$

Message Deadline Constraint: In our model, we assume that all the DYN messages must finish their transmissions before their deadlines. In the DYN segment, we can not get the schedule table for DYN messages, which means that it is hard to get the exact message transmission time. So we have to restrict that the worst response time for each DYN message should be smaller than or equal to the message deadline in Equation 2.28, then the scheduling of the DYN messages is guaranteed.

$$Rtime_m <= Mdeadline_m, \qquad m \in DYNmessage \tag{2.28}$$

2.2.2.3 Objective

The objective is represented as Equation 2.29, which is to minimize the worst response time for all the DYN messages in the system.

$$Minimize \sum_{m \in DYNmessage} Rtime_m \tag{2.29}$$

2.2.3 Case Study

In this section, a case study for DYN message scheduling optimization will be presented. The case study system contains two nodes with 5 DYN messages transmitting on a FlexRay communication bus as Figure 2.6. The data used by the case study about message sizes, the last minislot numbers and message deadlines are in Table 2.13.

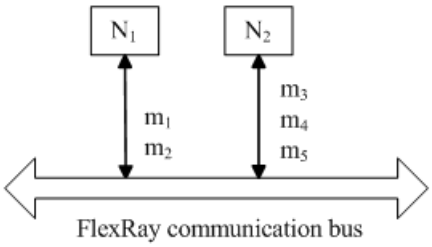


Figure 2.6: The architecture for the case study of the DYN message scheduling optimization

In this case study, all the DYN messages have fixed sizes as Table 2.13. We assume that each DYN message in the system should meet its deadline, and all the DYN messages are ready to transmit before the first DYN cycle. In order to find an optimization scheduling result which has the minimum total worst response time for all the DYN messages, we give a big enough deadline for each message as Table 2.13.

Node	Message	Size	pLatestTx	Deadline
n_1	m_1	5	7	100
n_1	m_2	3	9	100
n_2	m_3	4	8	100
n_2	m_4	6	6	100
n_2	m_5	7	5	100

Table 2.13: The data used for the case study of the DYN message scheduling optimization

Message	FrameID	roundBusCycle	Atime	Btime	Rtime
m_1	3	1	10	35	50
m_2	1	0	12	17	32
m_3	4	1	11	36	51
m_4	2	1	9	34	49
m_5	5	2	8	53	68

Table 2.14: The results for the case study of the DYN message scheduling optimization

In this case, we get a minimum value for the objective function, which is equal to 250 based on Equation 2.29. The result for this case is shown in Table 2.14.

Table 2.14 shows that m_2 gets the lowest frame identifier which means it has the highest priority for transmission, so it does not need to wait any cycle to transmit ($roundBusCycle_{m_2}$ is 0 in this case), and its worst response time is 32, which is the smallest one. m_5 gets the highest frame identifier, which means it has the lowest priority for its transmission. Because $roundBusCycle_{m_5}$ is 2, m_5 should wait at least two cycles to transmit.

If we change m_5's deadline from 100 to 40, then m_5 is supposed to be assigned to

Message	FrameID	roundBusCycle	Atime	Btime	Rtime
m_1	5	2	8	55	68
m_2	2	1	11	37	51
m_3	4	2	9	56	69
m_4	3	1	10	34	50
m_5	1	0	12	13	32

Table 2.15: The new results for the case study when m5's deadline is 40

Message	Deadline	FrameID
m_1	100	infeasible
m_2	100	infeasible
m_3	100	infeasible
m_4	100	infeasible
m_5	30	infeasible

Table 2.16: The Infeasible results when m5's deadline is 30

a lower frame identifer, because it has a earlier deadline now. In this new case, we get the new results in Table 2.15, and the objective for this new case is 270 based on Equation 2.29.

Table 2.15 shows that m_5 gets the lowest frame identifier which means it has the highest priority this time due to its earlier deadline. Correspondingly, it does not need to wait for any cycle to transmit because $roundBusCycle_{m_5}$ is 0, and its worst response time is only 32 in this case.

Table 2.14 and 2.15 show that 32 is the smallest worst response time among all the DYN messages in the case. If we change m_5's deadline from 40 to 30, we are supposed to get an infeasible result in this case. The results in Table 2.16 verify our assumption. In this case, m_5's deadline is too small to get a feasible frame identifier,

which leads this case to an infeasible result.

Chapter 3

Message Reliability Optimization

Introduction

In this chapter, we discuss how to increase the message reliability on the ground that each message is schedulable as discussed in Chapter 2. Due to FlexRay's flexible structure, it has more potential to support additional reliable requirements, and it will be used more and more in safety-critical applications. In order to ensure the reliability of message transmissions in FlexRay, we discuss two techniques to increase the reliability of message transmissions in 3.1 and 3.2, respectively.

3.1 Hardware Replication

The hardware replication technique is mostly used in many safety critical applications. In order to achieve the redundancy objective, the number of hardware components must be increased, such as channels and nodes.

3.1.1 Dual-channel

FlexRay uses two channels for message communications as TTP. In TTP, the communications on both channels are identical, which means one of the channels is used for the normal message communication, and the other one is employed for the message redundancy. Compared to TTP, FlexRay has more flexible choices. It can use one channel for normal traffic and the other for message redundancy as TTP in Figure 3.1. The other option is that we can transmit different messages on both channels to increase the bandwidth when there is no fault tolerance requirements needed in the application. In this case, two channels are independent for the message transmissions.

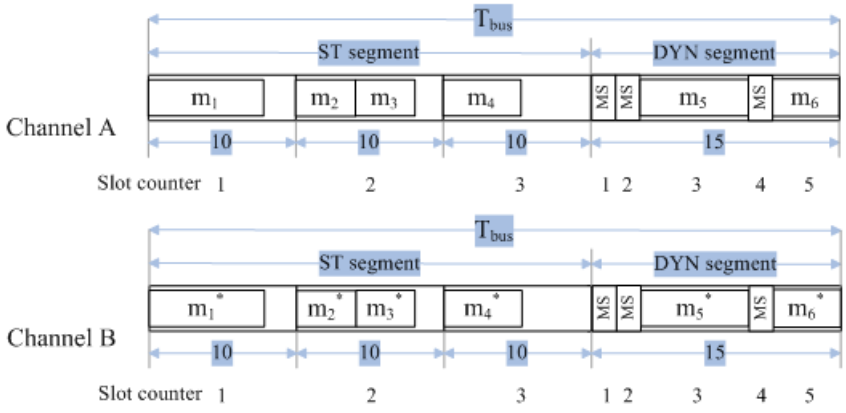


Figure 3.1: Dual-channel transmission example.

In Figure 3.1, messages m_1^*, $m_2^* \ldots m_6^*$ transmitted on channel B are the replicas of messages m_1, $m_2 \ldots m_6$ transmitted on channel A. (In the following content, m^* is always the replica message of m.) In this case, each message has one replica in order to increase the message reliability. It should be noticed that in this case, the whole channel B (including ST and DYN segments of each cycle) is used to transmit replica messages.

3.1.2 Fault-tolerant Unit

In order to tolerate node failures in the bus-based system, we can assign several nodes into a Fault-tolerant Unit (FTU) to provide redundancy services.

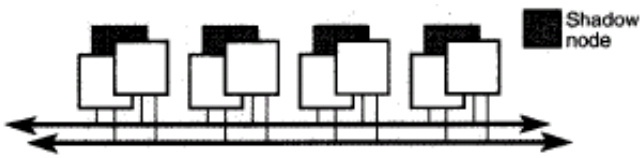


Figure 3.2: Fault-tolerant unit consisting of two active nodes and a shadow node from [12].

In Figure 3.2, there are four FTUs connected to the two bi-directional communication channels, and each FTU consists of two active nodes and a shadow node. Here, we assume these nodes are *fail-silent*[1]. When these two active nodes work correctly, the shadow node, which can be seen as a replicated node, only gets all the input messages, but does not transmit any output messages to the bus. When one of the active nodes fails, the shadow node will turn to be an active node and receive input and transmit output messages on two channels until the failed node comes back to the normal state[12]. FTU ensures that the redundancy mechanism can be started

[1] A fail-silent provides either correct results or no results at all[11]

within a short latency whenever an active node fails, and it can keep the system in the correct operation state during the repairing process of the failed node[11].

If nodes in the system are not fail-silent nodes, replications of nodes in the FTU described above are not enough to tolerate the node failure in the system. Usually it needs more hardware to implement the redundancy service. For example, if the failure happens in the communication network interface (CNI), a FTU needs three nodes and a voter to tolerate, which is called Triple-Modular redundancy. If *Byzantine failure*[2] occurs, which is one of the worst failures, the FTU must require at least four nodes and carry out a Byzantine-resilient agreement protocol to tolerate one Byzantine failure[11].

3.2 Re-execution

As discussed in Section 3.1.1, if both channels are independent, and messages transmitted on each channel are different, then there is no additional channel to ensure the fault tolerant requirements. In this section, we discuss how to use the re-execution technique to increase messages reliability on a FlexRay communication channel. That means even we transmit different messages on two channels, it can still satisfy fault tolerance requirements on both channels.

Each FlexRay cycle consists of a ST segment and a DYN segment as described in Chapter 2. The transmission in ST segments is similar with the communication in TTP cycle. They all use TDMA scheme to manage messages to access the bus. This scheme is very suitable for hard real-time messages due to its precision on the point-in-time. We can use ST segments for transmitting those hard real-time messages. The DYN segment is more flexible in time slot length. It can be used

[2]Fault node may provide different information to different observers[9].

for transmitting soft real-time messages or tolerating hard real-time messages to increase the messages reliability. Based on the above discussions, we illustrate two ways to increase the message reliability in 3.2.1 and 3.2.2, respectively.

3.2.1 With error notifications

If we have an error notification (ERRNOTIF) to describe which message has failed during its transmission, then we can just re-transmit those fault messages to increase the messages reliability. We assume that the ERRNOTIF is displayed in the first minislot of each DYN segment to check whether there are fault messages in the previous ST segment or not.

If a hard real-time message has errors during its transmission, it is indicated by the ERRNOTIF in the following DYN segment. Afterwards, the failed message needs to be re-transmitted when there is available space for its transmission. In Figure 3.3, the transmission of m_1 has failed in the ST segment, and m_1^* is re-transmitted in the DYN segment after the ERRNOTIF notices the transmission fault. Because m_1 has an early deadline, m_1^* is re-transmitted in the following DYN segment.

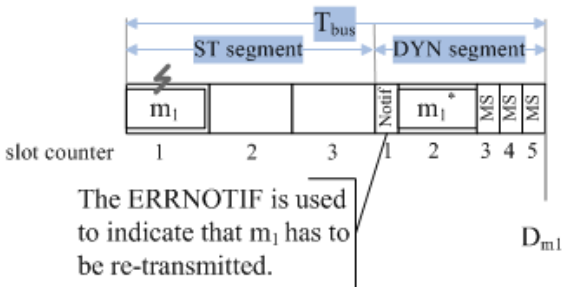


Figure 3.3: Re-transmit one message with the error notification (ERRNOTIF)

If there are more than one message needed to be re-transmitted, the value of the message priorities as Equation 3.1 should be compared. The message which has a

smaller value should be set a higher priority tab and be re-transmitted first.

$$P_m = D_m - R_{notif} - R_m \tag{3.1}$$

where P_m is the message priority value, D_m is the message deadline, R_{notif} is the response time of the ERRNOTIF, and R_m is the response time of the message m.

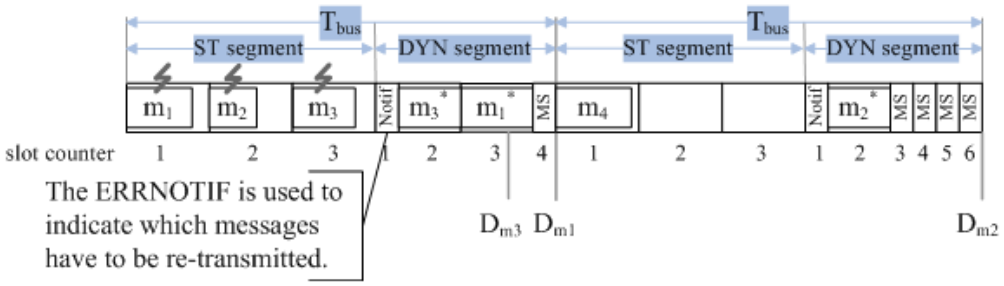


Figure 3.4: Re-transmit multiple messages with the error notification (ERRNOTIF)

In Figure 3.4, there are 4 hard real-time messages $m_1 \ldots m_4$ ready to transmit before the first cycle. Messages m_1, m_2 and m_3 are transmitted in the first, second and third ST slot of the first cycle, respectively. Message m_4 is transmitted in the first ST slot of the second cycle. Unfortunately, m_1, m_2 and m_3 have failed during their communication process. So after the error notification displayed in the first minislot of the first DYN segment, these three fault messages should be re-transmitted during the rest space of the DYN segment. Because $P_{m_3} > P_{m_1} > P_{m_2}$ based on Equation 3.1, m_3^* has a higher priority to transmit compared to m_1^* in the first cycle. There is not enough space for m_2^*'s re-transmission in the first DYN segment, thus m_2^* has to wait until the next DYN segment. We should know that if there are no redundant messages to be re-transmitted, then the DYN segment is used to transmit the soft real-time messages. Otherwise, those soft real-time messages may be delayed contributed by the transmission of the redundant messages.

The discussion above is an ideal case to increase the messages reliability. In this method, we assume that the ERRNOTIF in each DYN segment can indicate all the fault messages to nodes only within a very small period of time which is equal to the length of a minislot as Figure 3.3 and 3.4. Actually, it is not easy to ensure that all the redundant messages are ready to re-transmit just after the length of a minislot in each DYN segment. We need some configurations to make sure that at the beginning of the each DYN segment, the controller of each node will check if there are messages needed to be re-transmitted in the CHI buffer. If yes, the node has to write the redundant information in the ERRNOTIF. That means the ERRNOTIF needs to be updated at the beginning of each DYN segment.

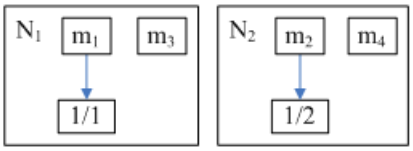
For example, in Figure 3.4, m_1^*, m_2^* and m_3^* are needed to be re-transmitted which is indicated by the ERRNOTIF at the beginning of the first DYN segment. Due to the limited segment length, only m_3^* and m_1^* are successfully re-transmitted in the first DYN segment. m_2^* is required to be re-transmitted when the ERRNOTIF is updated at the beginning of the second DYN segment.

Based on the above analysis, it is not easy to implement using the ERRNOTIF. If we give a priority to each message in advance, then we can just re-transmit the replicas for those hard real-time messages in order to increase the messages reliability in Section 3.2.2.

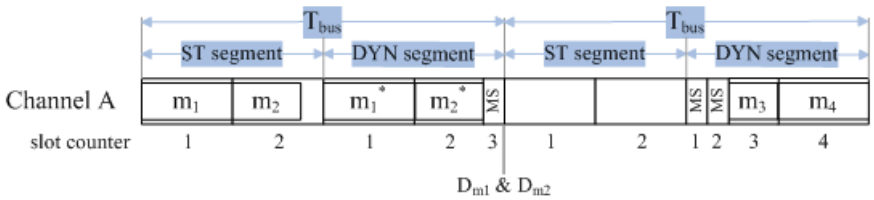
3.2.2 Without error notifications

In this section, whether the message fails during its transmission is unknown because there is no error notifications. We assume that there are two types of messages, hard real-time messages and soft real-time messages. In order to increase the messages reliability, we assume that each hard real-time message always has one replica even

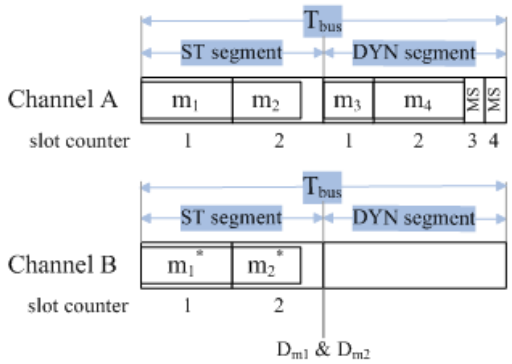
the original one does not have any fault. The replicas of the hard real-time messages can be transmitted in the ST or DYN segment by random. They can be also transmitted in either channel A or channel B depended on the message deadline.



(a) The node structure



(b) Re-transmission on the single channel



(c) Re-transmission on the dual-channel

Figure 3.5: Re-transmission example without the error notification.

In Figure 3.5(a), there are 2 nodes sending 4 messages $m_1 \ldots m_4$. We assume that m_1 and m_2 are hard real-time messages which means they have hard deadlines, and m_3 and m_4 are soft real-time message. In order to ensure that each hard real-time message can meet its deadline, all the hard real-time messages can only be

transmitted in the ST segment. Thus m_1 and m_2 are transmitted in the first and the second slot of the first cycle as Figure 3.5(b) and 3.5(c).

The assignment for the replicas are depended on the message deadline. In Figure 3.5(b), the deadlines of m_1 and m_2 are at the end of the first cycle, thus it is possible to re-transmit m_1^* and m_2^* in the following DYN segment of the same channel (Channel A) as Figure 3.5(b). If the message has a very early deadline, it may be only possible to re-transmit the replica on the different channels. The deadlines of m_1 and m_2 are at the end of the first ST segment in Figure 3.5(c). In order to ensure the message deadlines, m_1^* and m_2^* are re-transmitted in the ST segment of channel B.

Chapter 4

Implementation

Introduction

We have discussed the theoretical knowledge about how to schedule FlexRay messages and how to increase the messages reliability through the preceding chapters. In this chapter, we will explain our implementations based on the presented design. The proposed solutions are used to evaluate the performance of our implementations.

4.1 Design Overview

In this section, first, we discuss how to schedule FlexRay messages on two channels based on two different mechanisms for the ST and DYN message transmission. Second, we give a description on three message types used in our implementation. Last,

our design objective is explained.

4.1.1 Dual-channel Transmissions

We have already discussed how to schedule FlexRay messages on a FlexRay bus in Chapter 2. In order to assign FlexRay messages on both channel A and B, more variables, parameters and constraints are needed in our implementations.

For ST messages, two new sets *CHANNEL* and *CCS* are needed. The set *CHANNEL* is added to describe the available channels. The new set *CCS* is used to describe the available places for the message assignments. It replaces the old set *CS* described in Section 2.1.2. Using the set *CCS*, we not only decide the cycle and slot numbers for the ST messages as Section 2.1.2, but also determine the channel assignments for the ST messages.

CHANNEL	a set of channels
	CHANNEL = {A, B}
CCS	a set of channel-cycle-slot pairs representing the assigned
	place of the ST messages
	{(c, i, j)|c ∈ CHANNEL, i ∈ CYCLE, j ∈ SLOT}

Once the schedule table of the ST message is decided, no other factors can affect the message assignment. It is easy to assign the ST messages on both channel A and B. Actually, we do not care which channels the ST message is assigned to. In order to minimize the total response time for all the ST messages, we just need to make sure that the cycle number $Mcycle$ and the slot number $Mslot$ for each ST message are minimized in Equation 2.11.

For DYN messages, it is more complicated to decide the message assignments on two

channels. First, a new set *CHANNEL* and a new binary variable *CC* are needed. The set *CHANNEL* describes the available channels for the DYN messages. The variable *CC* is used to explain the relationship between any two DYN messages and the available channel in Equation 4.1.

$$CC_{i,j} = \begin{cases} 1, & \text{if DYN messages } i \text{ and } j \text{ are assigned on the same channel} \\ 0, & \text{otherwise} \end{cases} \tag{4.1}$$

Second, a new constraint in Equation 4.2 replaces the old one in Equation 2.23.

$$BusCycle_i = \sum_{j \in DYNmessage, i \neq j} MM_{i,j} * CC_{i,j} * Msize_j / pLatesTx_i, \quad i \in DYNmessage \tag{4.2}$$

In order to calculate the worst response time for the DYN message m, we need to know the messages which will delay the m's transmission on a FlexRay bus in the worst case. We use the binary variable *MM* and Equation 2.23 to describe how many cycles are delayed by those messages with lower frame identifiers in Section 2.2.2. If the DYN messages are assigned in two channels, both two variables *MM* and *CC* decide the *BusCycle* value as Equation 4.2. In this case, only those messages with lower frame identifiers which are transmitted on the same channel with the message m could delay the m's transmission, because the transmissions on channel A and B are independent. For example, if m_1 is transmitted on channel A and m_2 is transmitted on channel B, then m_1's transmission can not delay m_2's even m_1's frame identifier is lower than m_2's.

To calculate the result with ILOG CPLEX 10.0.0 optimization solver, only linear constraints are feasible. In Equation 4.2, we have the multiplication of two binary

variables, which means that it is a nonlinear expression. Nonlinear problems are much harder to solve than the comparable linear one, because in mathematic, it is harder to ensure that a nonlinear function of any number of variables is continuous and has a well-defined gradient at every point [16].

To solve this problem, we assume that the channel assignment of each DYN message is known in this thesis. We only need to decide the messages' frame identifers for those DYN messages which are transmitted on the same channel. For example, there are 5 DYN messages ready to transmit in the buffer. The channel assignments for the 5 messages are decided by the designer off-line. In this case, we assume that m_1 and m_2 are transmitted on channel A, and m_3, m_4 and m_5 are transmitted on channel B. So in our implementation, we just need to decide the message frame identifers for m_1 and m_2 on channel A, and the message frame identifers for m_3, m_4 and m_5 on channel B.

Two new sets $CDYNM$ and CF are added in the following content. The set $CDYNM$ replaces the old set $DYNmessage$ in Section 2.2.2. It describes the affiliations between channels and DYN messages. The set CF replaces the old set $FrameID$ in Section 2.2.2. It explains the affiliations between channels and frame identifers.

CDYNM	a set of channel-message pairs describing the affiliations between channels and messages $\{(c, m) \mid c \in$ CHANNEL, $m \in$ DYNmessage$\}$
CF	a set of channel-frame identifier pairs representing the affiliations between channels and frame identifiers {(c, fid)|c ∈ CHANNEL, fid ∈ FrameID}

Based on the set $CDYNM$, we know the number of DYN messages planning to transmit on the channel A and B. Then we can decide the number of frame identifiers

for each channel (the number of frame identifiers should be equal to the number of DYN messages on the same channel). Then the problem is simplified. We can just use the assignment mechanism discussed in Section 2.2.2 to calculate the worst response time of all the DYN messages on channels A and B, respectively.

4.1.2 Message Types

Critical, hard and soft messages are three message types for each node in our design.

Critical messages are very important messages in our application. All of them have hard deadlines. The whole system may crash if one of the critical messages fails to meet its deadline. Due to its importance, we assume that each critical message must have one replica message to increase the message reliability, such that the fault tolerant requirements are satisfied. In our design, both the critical message and the replica message must meet their deadlines as discussed in Section 3.2.2.

Hard messages also have hard deadlines as the critical messages, but they do not have any replicas compared to the critical messages. Each hard message must meet its hard deadline in our design.

Soft real-time messages do not have hard deadlines for their transmissions, and they do not need any replica messages either. We only need to ensure that each soft message has one chance to be transmitted on a FlexRay bus.

In the thesis, we use the term “original messages” to describe the three types of messages, critical, hard and soft messages, and we use the term “replica message” to describe the replica of the critical message.

As represented in Table 4.1, we assume that critical and hard messages can only be

Message type	Assignable segment
Critical messages	ST
Hard messages	ST
Soft messages	DYN
Replica messages	ST or DYN

Table 4.1: The message types and their assignable segments

assigned in ST segments due to their hard deadlines as the ST messages explained in Section 2.1. Soft messages are assigned in the DYN segments due to their flexible schedule as the DYN messages presented in Section 2.2. The replica messages can be transmitted in either ST or DYN segment decided by the designer off-line.

Based on the above discussions, several message sets are added in the program for our solutions in the following Section 4.2.1, 4.2.2 and 4.2.3.

- **New message sets**

criticalMESSAGE	a set of critical messages
hardMESSAGE	a set of hard messages
softMESSAGE	a set of soft real-time messages
STmessage	a set of ST message assigned in ST segments
	STmessage = criticalMESSAGE ∪ hardMESSAGE
DYNmessage	a set of DYN message assigned in DYN segments
	DYNmessage = softMESSAGE
MESSAGE	a set of messages
	MESSAGE = STmessage ∪ DYNmessage

4.1.3 Design Objective

The objective of our solutions in Section 4.2.1, 4.2.2 and 4.2.3 is to determine the schedule table for the critical, hard and a parts of replica messages, and decide the worst response time for the soft and the rest part of replica messages based on the knowledge in Chapter 2, such that all the messages are schedulable, and the total response time for all the messages is minimized.

4.2 Implementation Solutions

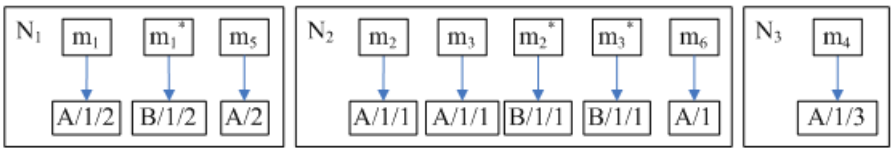
Three solutions are explained according to the above discussion. One is the naive solution, another is the optimal solution with replicas, and the other is adding redundancy to the optimal solution without replicas. The first two solutions are required to satisfy fault tolerant requirements. In these two solutions, the transmissions of the replica messages must be ensured.

4.2.1 Naive Solution

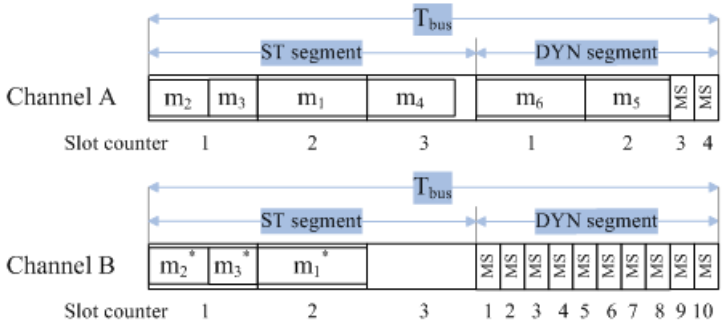
We call it "the naive solution", because in this solution, we just need a simple copy process for critical messages from channel A to channel B.

In this case, all the original messages are transmitted on channel A which is used for normal transmissions. All the replica messages are transmitted in ST segments of channel B which is only used for redundancy.

There are 3 nodes sending 6 messages in Figure 4.1(a). m_1, m_2 and m_3 are critical messages. m_4 is a hard message. m_5 and m_6 are soft messages. The schedule



(a) The message schedule tables and the relationship between nodes and messages



(b) Message assignment for the naive solution.

Figure 4.1: An example for the naive solution.

table for ST messages ($m_1 \ldots m_4$) and the assignments of frame identifiers for DYN messages (m_5 and m_6) are decided by the program in Appendix A.3. All the original messages $m_1 \ldots m_6$ are assigned on channel A. The replica messages m_1^*, m_2^* and m_3^* are assigned on channel B to increase the reliability of the critical messages as Figure 4.1(b).

The assignments of replica messages on channel B are the same as the assignments of the corresponding critical messages on channel A. For example, in Figure 4.1, the critical message m_1 is transmitted in the second slot of the first cycle on channel A. Then the corresponding replica message m_1^* must be transmitted in the second slot of the first cycle on channel B.

It is noticeable that we can not get a schedule table for the DYN messages, because the program can only decide the worst response time for DYN messages. So the

assignments of m_5 and m_6 in Figure 4.1(b) are based on the specification in [1] and the description in Section 2.2.1.

Actually, in the naive solution, we only need to consider how to schedule the original messages on a single channel (channel A). The objective is to minimize the original messages response time on channel A and the replica messages response time on channel B as Equation 4.3. The program of the naive solution is in Appendix A.3 based on a combination of the modeling description in Section 2.1.2 and 2.2.2.

$$Minimize \sum_{m \in MESSAGE} Rtime_m + \sum_{m \in criticalMESSAGE} Rtime_m \tag{4.3}$$

4.2.2 Optimal Solution With Replicas

Different with the naive solution in Section 4.2.1, both channel A and B are used for the normal transmission in the optimal solution with replicas, which means channel A and B are independent and the original and replica messages can be transmitted on the two channels.

In this case, each critical message also has one replica to increase its reliability. We must ensure that both the critical message and its replica meet the deadline. For example, the critical message m_1 and its replica m_1^* must finish their transmissions before m_1's deadline in Figure 4.2.

As represented in Table 4.1, the replica messages can be assigned in either ST or DYN segments in this solution. For simplicity, the replica messages assigned in ST or DYN segments is decided by the data file as shown in Appendix A.4.

There are several additional sets in addition to the new message sets presented in

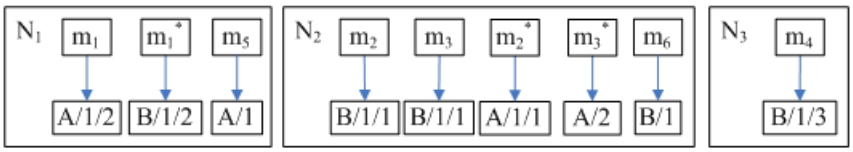
Section 4.1.2. The ST messages contain critical, hard messages and a part of replica messages. The DYN messages includes soft messages and the rest part of replicas messages.

- **Additional sets**

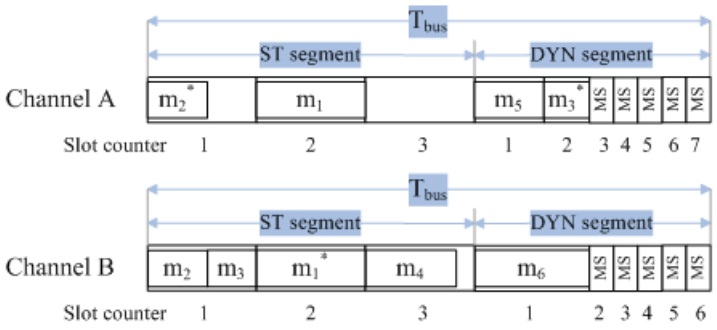
replicaInST	a set of replicas belonging to the ST messages
replicaInDYN	a set of replicas belonging to the DYN messages
STmessage	a set of ST messages STmessage = criticalMESSAGE ∪ hardMESSAGE ∪ replicaInST
DYNmessage	a set of DYN messages DYNmessage = softMESSAGE ∪ replicaInDYN
CHANNEL	a set of channels CHANNEL = {A, B}
CCS	a set of channel-cycle-slot pairs representing ST messages schedule table {(c, i, j)|c ∈ CHANNEL, i ∈ CYCLE, j ∈ SLOT}

We use the same example presented in the naive solution to describe how the messages are assigned using the optimal solution with replicas as Figure 4.2. In this case, we assume that the replica messages m_1^* and m_2^* are assigned in the ST segment as two ST messages, and m_3^* is assigned in the DYN segment as a DYN message. For the three DYN messages, m_5, m_6 and m_3^*, we assume that m_5 and m_3^* are assigned on channel A, and m_6 is assigned on channel B.

We can see that there are original and replica messages transmitted on both channel A and B in Figure 4.2, and the two channels are independent for message transmissions in this case.



(a) The message schedule tables and the relationship between nodes and messages



(b) Message assignment for the optimal solution with replicas

Figure 4.2: An example for the optimum solution with replicas

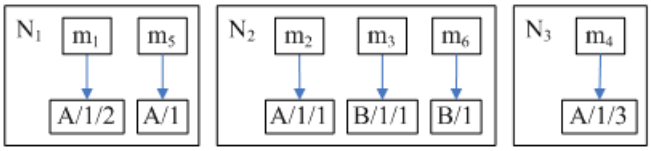
4.2.3 Adding Redundancy to the Optimal Solution Without Replicas

In the optimal solution without replicas, critical messages do not have any replicas. That means we just need to decide the solution for all the original messages without replicas on the two channels. We can get an optimal solution for the assignments of all the original messages, but in this case it can not satisfy the fault tolerant requirements for safety critical applications. In the following content, we discuss how to add redundancy functions for this solution, such that it is possible to ensure the fault tolerant requirements to increase the message reliability.

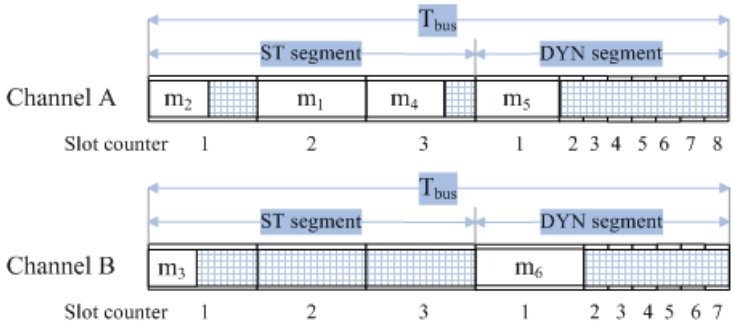
The same with the optimal solution with replicas, the original message are transmitted on both channel A and B in this solution. So two new set $CHANNEL$ and CCS are also needed as Section 4.2.2. The other message sets for this solution are the same as the description in Section 4.1.2. The different place is that the replica messages are only assigned in the rest of the available space after the assignments of all the original messages are decided.

We use the same example presented in the naive solution to describe how the original messages are assigned using this solution as Figure 4.3. First, we assign the original messages on both channel A and B, then the redundancy messages (replica messages) can be put in the shadow space to increase the message reliability as Figure 4.3(b).

We noticed that the assignments of the original messages must be first ensured in this solution. If there is enough shadow space for the replica messages transmissions, then the fault tolerant requirements can be ensured. Otherwise, some replica messages may not finish their transmissions before their deadlines. In other words, this solution could not ensure all the original and replica messages transmissions for safety critical applications.



(a) The relationship between nodes and messages



(b) The optimum solution without replicas for the original messages assignments and the available space for the replica messages.

Figure 4.3: An example for the optimum solution without replicas.

Chapter 5

Evaluation

Introduction

The idea of the evaluation is to analyse the performance of the different solutions discussed in Chapter 4. The performance includes the execution time and the objective value obtained from the solutions. In Section 5.1, we apply the evaluation analysis for the example discussed in Chapter 4, and compare the results based on the two different solutions: the naive solution and the optimal solution with replicas. A real-world example is also used to evaluate these two optimization solutions in Section 5.2.

5.1 Evaluation Analysis

In this section, two solutions, NS and OS^+, are to analyse and evaluate using the example shown in Figure 4.1 and 4.2. The experiments are performed on a laptop with Intel Pentium processor 1.5GHz and 768 MB RAM.

Abbreviations	Solutions
NS	Naive solution (4.2.1)
OS^+	Optimum solution with replicas (4.2.2)

In this example, we generate 6 messages mapped to the system architecture consisting of 3 nodes. It has 3 critical messages, 1 hard message and 2 soft messages. The affiliations between nodes and messages are shown in Figure 4.1(a) and 4.2(a). The results based on the two solutions are displayed in Table 5.1. The objective value is the total response time of all the messages. The MIP simplex iterations controls the number of simplex iterations performed on each variable [21].

As shown in Table 5.1, the objective value of OS^+ (179) is bigger than that of NS (149). We can see that the response time of m_3^* using OS^+ (59) is much bigger than using NS (5). The difference value is because in OS^+, we assume that m_3^* is assigned in the DYN segment as a DYN message, and in NS, m_3^* is seen as a ST message which is transmitted in the ST segment. For each DYN message, we can only get the worst response time, which may be much bigger than the actual message response time. Thus when we assume m_3^* is a ST message in OS^+, we can get a smaller objective value which is only equal to 125.

	NS	OS^+
$Rtime_{m_1}$	10	10
$Rtime_{m_2}$	5	5
$Rtime_{m_3}$	5	5
$Rtime_{m_4}$	15	15
$Rtime_{m_5}$	59	35
$Rtime_{m_6}$	35	35
$Rtime_{m_1^*}$	10	10
$Rtime_{m_2^*}$	5	5
$Rtime_{m_3^*}$	5	59
Objective	149	179
MIP simplex iterations	14	30

Table 5.1: The Results of the example as shown in Figure 4.1 and 4.2 using NS and OS^+

5.2 A Real-life Example

In order to evaluate our solutions on a real-world example, we use the real-life example implementing a vehicle cruise controller from [2]. It consists of 26 messages mapped to 5 nodes. We assume that there are 6 critical messages, 12 hard messages and 8 soft messages, and each critical message must have one replica to increase the message reliability. So actually, we should schedule 32 (26 original messages + 6 replica messages) messages in total in this example. However, due to the missing information about the message sizes in this example, we randomly generate several integers in a specified range as the message sizes.

Performance	NS	OS^+
Objective	3596	3166
MIP simplex iterations	5478578	1318528
Branch-and-bound nodes	586254	237849
Execution time	11min	3min

Table 5.2: The performances of the real-life example using NS and OS^+

Table 5.2 shows that OS^+ produces a smaller (12%) objective value and a shorter (73%) execution time than NS in the real-life example. Scheduling the example using NS takes about 11 minutes, while OS^+ is faster, finishing all the message scheduling in only 3 minutes. The example using NS gives 3596 as all the message response time, while OS^+ produces a smaller value which is only equal to 3166.

Figure 5.1 and 5.2 show the message assignments using NS and OS^+, respectively. We can see that it takes 4 communication cycles to schedule all the original and replica messages by using NS. While using OS^+, it only takes 2 communication cycles. In OS^+, the original and replica messages can be assigned in both channel A and B as Figure 5.2, which means the space on these two channels are used sufficiently

for all the message transmissions. In NS, the original messages are assigned on the whole channel A, including both ST and DYN segments. The replica message are only assigned in the ST segments of channel B. So actually, the space in the DYN segments of channel B are useless as shown in Figure 5.1. That is why scheduling the example using NS needs more communication cycles than using OS^+.

FrameID	1	2	3	4	5	6	7	8
DYN message	m_{20}	m_{25}	m_{19}	m_{22}	m_{21}	m_{26}	m_{23}	m_{24}
roundBusCycle	0	1	1	1	1	2	2	3
The actual response time	93	96	98	100	101	202	304	408
The worst response time	114	215	214	213	212	313	312	413

Table 5.3: The results for the DYN messages in the real-life example using NS.

FrameID	A/1	A/2	A/3	A/4	A/5	B/1	B/2	B/3	B/4	B/5
DYN message	m_{23}	m_{20}	m_{19}	m_{22}	m_{21}	m_6^*	m_{25}	m_{24}	m_5^*	m_{26}
roundBusCycle	0	1	1	1	1	0	1	1	1	2
The actual response time	94	97	99	101	102	91	94	99	102	201
The worst response time	114	215	214	213	212	114	215	214	213	314

Table 5.4: The results for the DYN messages in the real-life example using OS^+.

There are 8 soft real-time messages in this example. In NS, each soft message is seen as a DYN message corresponding to an unique frame identifer as Table 5.3. In OS^+, we assume that two replica messages m_5^* and m_6^* are assigned in the DYN segments as two DYN messages. So there are 10 messages assigned in the DYN segments in all. We assume that there are 5 DYN messages transmitted on both channel A and B corresponding to an unique frame identifer as Table 5.4.

Table 5.3 and 5.4 show that the worst response time for each DYN message is big

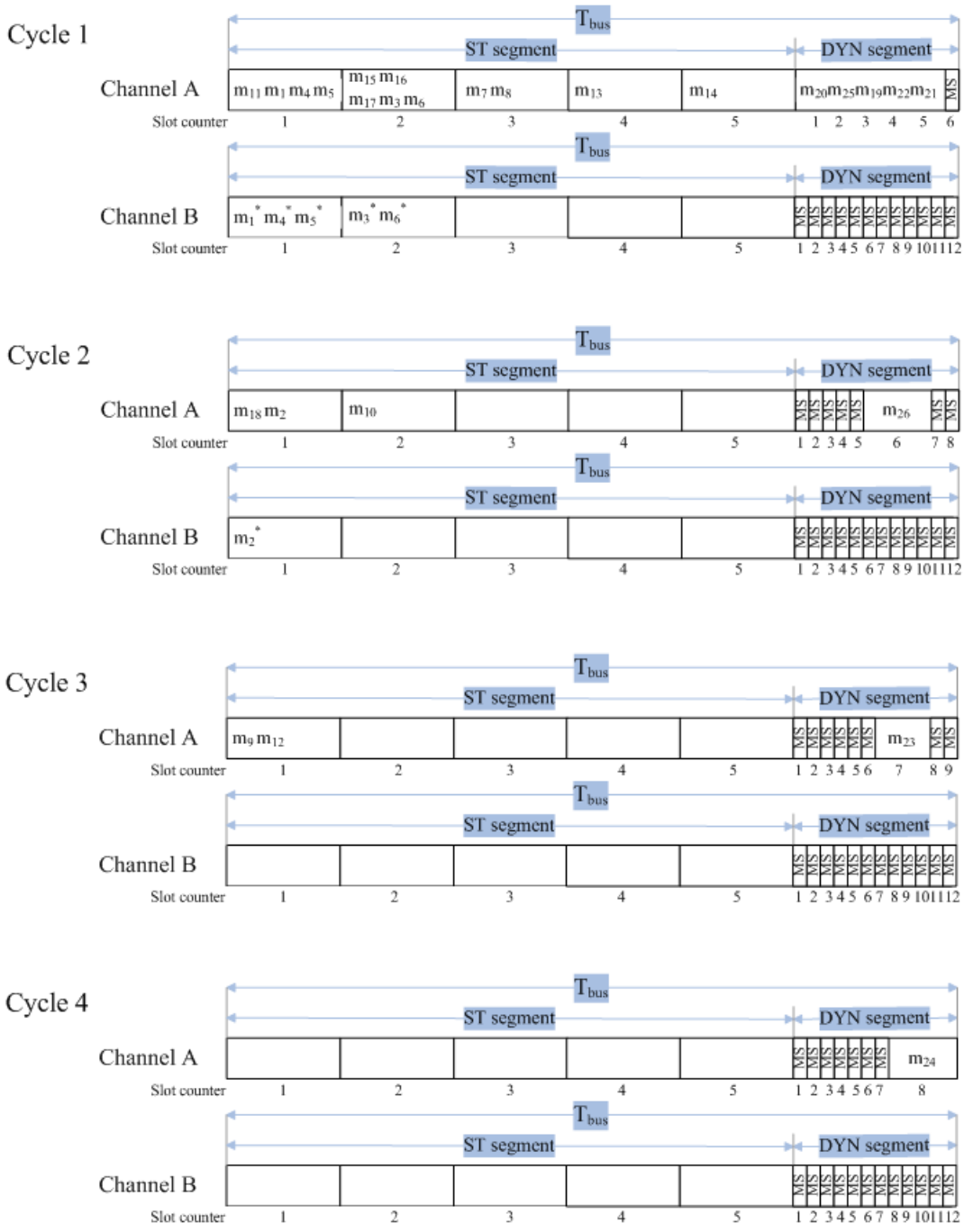


Figure 5.1: The message assignment of the real-life example using NS

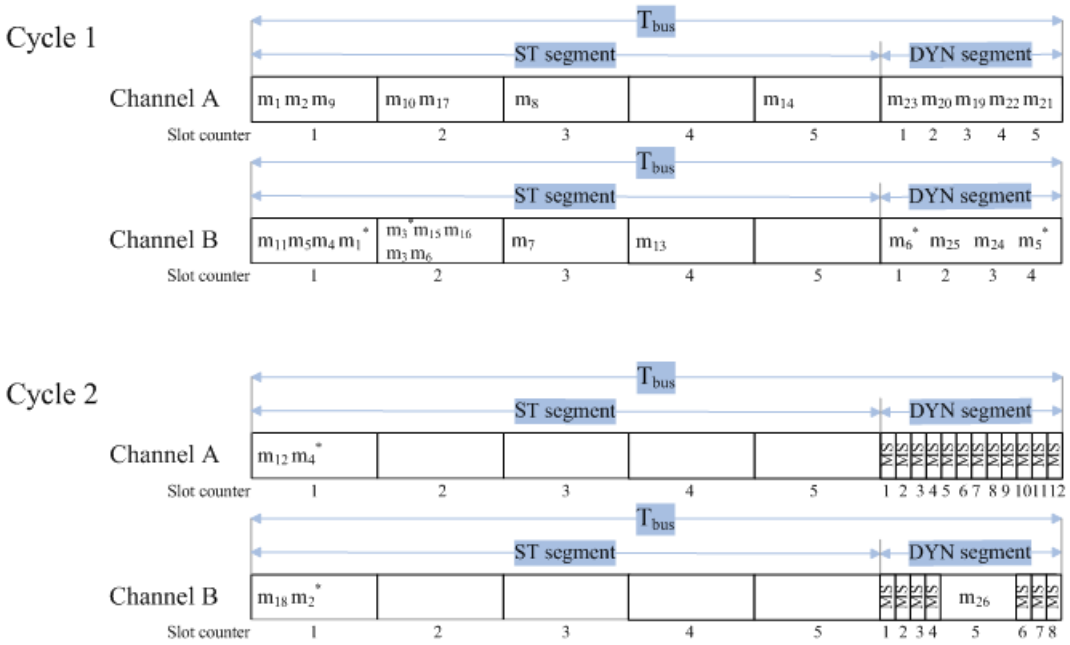


Figure 5.2: The message assignment of the real-life example using OS^+

enough compared to the actual response time, which means the worst response time calculated by the two solutions are reasonable. We can use these two solutions to check if the message deadline is met or not before starting the DYN message transmission in a real-world example.

Chapter 6

Conclusion

In this thesis, we have determined two solutions which combined several redundancy techniques for the FlexRay message scheduling in an application, such that the message transmission meet the reliability purpose, and the application is schedulable.

We have described how messages are transmitted in ST and DYN segments of a FlexRay bus. Based on two different bus access schemes in the ST and DYN segment, we discussed two optimization approaches to model message scheduling, respectively. For ST messages, we implemented a schedule table for all the messages, and ensured that each message can finish its transmission before its deadline. For DYN message, we gave an unique frame identifer to each message, and calculated the message worst response time. Based on the worst response time, we can check if the DYN message deadline was met or not. For both ST and DYN messages, our objective of these two optimization approaches was minimized the total response time for all the messages.

Since FlexRay will be used more and more in safety-critical applications, the message transmission has to be reliable. We have investigated two techniques, the hardware replication and the re-execution, to increase the message reliability on the ground that each FlexRay message is schedulable. For the replication technique, we increased the number of hardware components, such as channels and nodes, to achieve the redundancy purpose. For the re-execution technique, we considered to use the DYN segments to re-transmit hard real-time messages for increasing the message reliability. Two methods, with error notification and without error notification, which dealt with the message re-transmission were also discussed.

Finally, we have proposed two solutions to satisfy fault tolerant requirements. NS uses a simple copy process for critical messages from channel A to channel B. OS^+ allows the original and replica messages to transmit on both channel A and B based on the without error notification method. We use a real-life example to evaluate these two solutions. Our experiments have shown that the OS^+ has better performances, which produces a smaller (12%) objective value and a shorter (73%) execution time than NS.

Bibliography

[1] *FlexRay Communications System Protocol Specification Version 2.1 Revision A*, 22-December-2005.

[2] T. Pop, P. Pop, P. Eles, Z. Peng, A. Andrei. *Timeing Analysis of the FlexRay Communication Protocol*, Springer Science+Business Media, LLC 2007.

[3] T. Pop, P. Pop, P. Eles, Z. Peng. *Bus Access Optimisation for FlexRay-based Distributed Embedded System*, 2007.

[4] R. Makowitz. *FlexRay - A Communication Network for Automotive Control Systems*, IEEE 2006.

[5] J. Rushby. *Bus Architectures for Safety-Critical Embedded Systems.*

[6] H. Kopetz. *A Comparison of TTP and FlexRay*, Techinical University of Wien, Austria, 2001.

[7] L. M. Pinho, F. Vasques. *Reliable Communication in Distributed Computer-Controlled Systems*, Ada-Europe 2001, LNCS 2043, pp. 136-147, 2001.

[8] *http://en.wikipedia.org/wiki/Vehicle_bus.*

[9] C. Ryan, D. Heffernan, G. Leen. *Additional Communication System Services for Safety-Critical Applications*, ISSC 2005, Dublin, September 1-2.

[10] R. Johansson. *Time and event triggered communication scheduling for automotive applications*, Chalmers Lindholmen University, 2004.

[11] H. Kopetz. *Real-time Systems: Design Principles for Distributed Embedded Applications*, Chapter 6, Springer.

[12] H. Kopetz, G. Grünsteidl. *TTP-A protocol for Fault Tolerant Real-Time Systems*, 1994 IEEE.

[13] J. Cosgrove and B. Donnelly. *Intelligent automotive networks for distributed real-time control*, 2003.

[14] Dr. Christopher Temple. *FlexRay International Workshop*, 2003.

[15] P. M. Szecowka, M. A. Swiderski. *On Hardware Implementation of FlexRay Bus Guardian Module*, 2007.

[16] R. Fourer, David M. Gay, Brain W. Kernighan. *AMPL—A Modeling Language for Mathmatical Programming*, 1993.

[17] H. Paul Williams. *Model Building in Mathematical Programming*, 1999.

[18] K. Sandström, C. Norström, M. Ahlmark. *Frame Packing in Real-Time Communication*, 2000 IEEE.

[19] B. W. Johnson. *An Introduction to the Design and Analysis of Fault-Tolerant Systems*, 1989.

[20] B. Rogers, S. Schmechtig. *FlexRay message buffers*, 2006.

[21] ILOG. *ILOG CPLEX 6.5 Reference manual*, 1999.

[22] *http://en.wikipedia.org/wiki/Branch-and-bound*

[23] N. Navet, Y. Song, F. Simonot-lion, C. Wilwert. *Trends in automotive communication systems*, 2005.

[24] H.Kopetz. *A Comparison of CAN and TTP*, Annual Reviews in Control 24 (2000) 177-188.

[25] G. Cena, A. Valenzano. *Performace Analysis of Byteflight Networks*, Proceedings of the IEEE International Workshop on Factory Communication Systems, pp 157-166, 2004.

[26] S. Ding, N. Murakami, H. Tomiyama, H. Takada. *A GA-Based Scheduling Method for FlexRay Systems*, 2005.

Appendix A AMPL Programs

A.1 ST Messages Scheduling Optimization

```
#### The Beginning of the Model file ####

### Sets ###

 set STmessage;
 set NODE;
 set NM within ( NODE cross STmessage );
 set CYCLE;
 set SLOT;
 set CS within (CYCLE cross SLOT);

### Parameters ###

 param Sslot;
 param STsize := Sslot * card(NODE);
```

```
 param DYNsize;
 param Tbus := STsize + DYNsize ;

 param Msize {STmessage};
 param Mdeadline {STmessage};

### Variables ###

 var message_assign{CS, NM} binary;
 var node_assign{SLOT, NM} binary;

 var Rtime{STmessage};
 var Mcycle{STmessage};
 var Mslot{STmessage};


### Objective ###

 minimize Obj: sum{m in STmessage} Rtime[m];


## Message Mutual Exclusion Constraint ##

subject to c1{(n,m) in NM}:
           sum {(c,s) in CS} message_assign[c,s,n,m] = 1 ;

subject to c2{(n,m) in NM}:
           sum{s in SLOT} node_assign[s,n,m] = 1;
```

```
### Message, Node and Slot Constraints ###

subject to c3{s in SLOT,(n,p) in NM,(n,q) in NM}:
           node_assign[s,n,p] = node_assign[s,n,q];

subject to c4{s in SLOT}:
           sum{(n,m) in NM} node_assign[s,n,m] >= 1;

subject to c5{(c,s) in CS}:
           sum{(n,m) in NM} message_assign[c,s,n,m] * Msize[m]
                            <= Sslot ;

subject to c6{s in SLOT, n in NODE}:
           sum{(c,s) in CS, (n,m) in NM} message_assign[c,s,n,m]
         = sum{(n,m) in NM} node_assign[s,n,m];


### Message Deadline Constraints ###

subject to c7{m in STmessage}:
           Mcycle[m] = sum{(c,s) in CS} message_assign[c,s,n,m] * c ;

subject to c8{m in STmessage}:
           Mslot[m] = sum{(c,s) in CS} message_assign[c,s,n,m] * s;

subject to c9{m in STmessage}:
```

```
          Rtime[m] = (Mcycle[m] - 1) * Tbus + Mslot[m] * Sslot;

subject to c10{m in STmessage}:
          Rtime[m] <= Mdeadline[m];

#### The End of the Model file ####


#### The Beginning of the Data file ####

 set MESSAGE :=  m1  m2  m3  m4  m5  m6  m7  m8  m9;
 set NODE := n1  n2  n3;
 set NM :=
    (n1, *) m1  m2  m3  m4
    (n2, *) m5  m6
    (n3, *) m7  m8  m9        ;

 set CYCLE := 1 2 3 4 5;
 set SLOT := 1 2 3;
 set CS :=
     (1, *) 1 2 3
     (2, *) 1 2 3
     (3, *) 1 2 3
     (4, *) 1 2 3
     (5, *) 1 2 3 ;

 param Sslot := 5;
 param DYNsize := 15;
```

```
param:        Mdeadline   :=
      m1        40
      m2        60
      m3        70
      m4        90
      m5        20
      m6        50
      m7        50
      m8        70
      m9        90        ;

param:        Msize   :=
      m1        5
      m2        2
      m3        4
      m4        3
      m5        4
      m6        2
      m7        1
      m8        4
      m9        4     ;

#### The End of the Data file ####
```

A.2 DYN Messages Scheduling Optimization

```
#### The Beginning of the Model file ####

### Sets ###

 set DYNmessage;
 set FrameID;

     check card(DYNmessage) = card(FrameID);

### Parameters ###

 param STsize;
 param DYNsize;
 param Tbus = STsize + DYNsize;
 param gdMinislot;

 param Msize{DYNmessage};
 param Mdeadline{DYNmessage};
 param pLatestTx{m in DYNmessage} = DYNsize - Msize[m];

 param bignum;

### Variables ###

 var message_to_frameID{FrameID,DYNmessage} binary;
 var Mframeid{DYNmessage} in FrameID;
```

```
 var MM{i in DYNmessage, j in DYNmessage : i<>j} binary;
 var TT{i in DYNmessage, j in DYNmessage : i<>j} binary;

 var BusCycle{DYNmessage};
 var roundBusCycle{DYNmessage} integer;
 var Rtime{DYNmessage};
 var Atime{DYNmessage};
 var Btime{DYNmessage};


### Objective ###

 minimize Totaltime:  sum{m in DYNmessage} Rtime[m] ;


### Constraints ###

 subject to c1{fid in FrameID}:
            sum{m in DYNmessage} message_to_frameID[fid,m] = 1;

 subject to c2{m in DYNmessage}:
             sum{fid in FrameID} message_to_frameID[fid,m] = 1;

 subject to c3{m in DYNmessage}:
             Mframeid[m] =
              sum{fid in FrameID} message_to_frameID[fid,m] * fid ;
```

```
subject to c4{i in DYNmessage, j in DYNmessage : i<>j}:
            Mframeid[i] - Mframeid[j] <= bignum * MM[i,j] ;

subject to c5{i in DYNmessage, j in DYNmessage : i<>j}:
            Mframeid[j] - Mframeid[i] <= bignum * TT[i,j] ;

subject to c6{i in DYNmessage, j in DYNmessage : i<>j}:
            MM[i,j] + TT[i,j] = 1;

subject to c7{i in DYNmessage}:
            BusCycle[i] = sum{j in DYNmessage:i<>j}
                            MM[i,j] * Msize[j] / pLatestTx[i];

subject to c8{m in DYNmessage}:
            roundBusCycle[m] >= BusCycle[m];

subject to c9{m in DYNmessage}:
           Atime[m] = Tbus - STsize - (Mframeid[m] - 1) * gdMinislot;

subject to c10{m in DYNmessage}:
           Btime[m] = roundBusCycle[m] * Tbus
                      + STsize + pLatestTx[m] * gdMinislot;

subject to c11{m in DYNmessage}:
           Rtime[m] = Atime[m] + Btime[m] + Msize[m];

subject to c12{m in DYNmessage}:
           Rtime[m] <= Mdeadline[m];
```

```
#### The End of the Model file ####

#### The Beginning of the Data file ####

 set DYNmessage :=  m1  m2  m3  m4  m5;
 set FrameID := 1 2 3 4 5;

 param STsize := 8;
 param DYNsize := 12;
 param gdMinislot := 1;
 param bignum := 1000;

 param:       Msize   :=
       m1       5
       m2       3
       m3       4
       m4       6
       m5       7     ;

 param:       Mdeadline   :=
       m1       100
       m2       100
       m3       100
       m4       100
       m5        40   ;

#### The End of the Data file ####
```

A.3 Naive Solution

```
#### The Beginning of the Model file ####

### Sets ###

 set criticalMESSAGE;
 set hardMESSAGE;
 set softMESSAGE;

 set STmessage := criticalMESSAGE union hardMESSAGE;
 set DYNmessage := softMESSAGE;
 set MESSAGE := STmessage union DYNmessage;

 set NODE;
 set STNM within ( NODE cross STmessage );
 set FrameID;

      check card(DYNmessage) = card(FrameID);

 set CYCLE;
 set SLOT;
 set CS within (CYCLE cross SLOT);
```

```
### Parameters ###

 param Msize{MESSAGE};
 param Mdeadline{STmessage};

 param Sslot;
 param STsize := card(NODE) * Sslot;
 param DYNsize;
 param Tbus := STsize + DYNsize ;

 param gdMinislot;
 param pLatestTx{m in DYNmessage} := DYNsize - Msize[m] ;

 param bignum;


### Variables ###

 var message_assign{CS, STNM} binary;
 var node_assign{SLOT, STNM} binary;

 var Mcycle{STmessage};
 var Mslot{STmessage};

 var message_to_frameID{FrameID,DYNmessage} binary;
 var Mframeid{DYNmessage} in FrameID;

 var MM{i in DYNmessage, j in DYNmessage : i<>j} binary;
```

```
 var TT{i in DYNmessage, j in DYNmessage : i<>j} binary;

 var BusCycle{DYNmessage};
 var roundBusCycle{DYNmessage} integer;
 var Atime{DYNmessage};
 var Btime{DYNmessage};
 var Rtime{MESSAGE};


### Objective ###
 minimize Obj:   sum{m in MESSAGE} Rtime[m]
               + sum{m in criticalMESSAGE} Rtime[m];


###------ The beginning for Constraints of ST messages ------- ###

## Message Mutual Exclusion Constraint ##

 subject to c1{(n,m) in STNM}:
           sum {(c,s) in CS} message_assign[c,s,n,m] = 1 ;

 subject to c2{(n,m) in STNM}:
           sum{s in SLOT} node_assign[s,n,m] = 1;


### Message, Node and Slot Constraints ###

 subject to c3{s in SLOT,(n,p) in STNM, (n,q) in STNM: p<>q}:
```

```
          node_assign[s,n,p] = node_assign[s,n,q];

 subject to c4{s in SLOT}:
          sum{(n,m) in STNM} node_assign[s,n,m] >= 1;

 subject to c5{(c,s) in CS}:
          sum{(n,m) in STNM}
             message_assign[c,s,n,m] * Msize[m] <= Sslot ;

 subject to c6{s in SLOT, n in NODE}:
          sum{(c,s) in CS, (n,m) in STNM} message_assign[c,s,n,m]
        = sum{(n,m) in STNM} node_assign[s,n,m];

### Message Deadline Constraints ###

 subject to c7{(n,m) in STNM}:
          Mcycle[m] = sum{(c,s) in CS} message_assign[c,s,n,m] * c ;

 subject to c8{(n,m) in STNM}:
          Mslot[m] = sum{(c,s) in CS} message_assign[c,s,n,m] * s;

 subject to c9{m in STmessage}:
          Rtime[m] = (Mcycle[m] - 1) * Tbus + Mslot[m] * Sslot;

 subject to c10{m in STmessage}:
          Rtime[m] <= Mdeadline[m];

 subject to SlotSizeConstraint:
```

```
          Sslot >= max{m in STmessage} Msize[m];

###------ The end for Constraints of ST messages ------- ###


###------ The beginning for Constraints of DYN messages ------- ###

 subject to c11{fid in FrameID}:
            sum{m in DYNmessage} message_to_frameID[fid,m] = 1;

 subject to c12{m in DYNmessage}:
            sum{fid in FrameID} message_to_frameID[fid,m] = 1;

 subject to c13{m in DYNmessage}:
             Mframeid[m] =
                sum{fid in FrameID} message_to_frameID[fid,m] * fid ;

 subject to c14{i in DYNmessage, j in DYNmessage : i<>j}:
             Mframeid[i] - Mframeid[j] <= bignum * MM[i,j] ;

 subject to c15{i in DYNmessage, j in DYNmessage : i<>j}:
             Mframeid[j] - Mframeid[i] <= bignum * TT[i,j] ;

 subject to c16{i in DYNmessage, j in DYNmessage : i<>j}:
             MM[i,j] + TT[i,j] = 1;

 subject to c17{i in DYNmessage}:
             BusCycle[i] = sum{j in DYNmessage:i<>j}
```

```
                              MM[i,j] * Msize[j] / pLatestTx[i];

 subject to c18{m in DYNmessage}:
             roundBusCycle[m] >= BusCycle[m];

 subject to c19{m in DYNmessage}:
            Atime[m] = Tbus - STsize - (Mframeid[m] - 1) * gdMinislot;

 subject to c20{m in DYNmessage}:
            Btime[m] =  roundBusCycle[m] * Tbus + STsize
                      + pLatestTx[m] * gdMinislot;

 subject to c21{m in DYNmessage}:
            Rtime[m] = Atime[m] + Btime[m] + Msize[m];

 subject to DYNsizeConstraint:
            DYNsize >=   max{m in DYNmessage} Msize[m]
                       + (card(FrameID) - 1) * gdMinislot;


###------ The end for Constraints of DYN message ------- ###


#### The End of the Model file ####


#### The Beginning of the Data file ####
```

```
set criticalMESSAGE :=  m1  m2  m3;
 set hardMESSAGE := m4;
 set softMESSAGE := m5  m6;

 set NODE := n1  n2  n3;
 set STNM :=
    (n1, *) m1
    (n2, *) m2  m3
    (n3, *) m4 ;

 set CYCLE := 1 2 3 ;
 set SLOT := 1 2 3;
 set CS :=
    (1, *) 1 2 3
    (2, *) 1 2 3
    (3, *) 1 2 3;

 set FrameID := 1 2;

 param Sslot := 5;
 param DYNsize := 10;
 param gdMinislot := 1;
 param bignum := 1000;

 param:       Msize   :=
       m1       5
       m2       3
       m3       2
```

```
      m4       4
      m5       3
      m6       5 ;

 param:       Mdeadline   :=
      m1       100
      m2       100
      m3       100
      m4       100     ;

#### The End of the Data file ####
```

A.4 Optimum Solution with Replicas

```
#### The Beginning of the Model file ####

### Sets ###

 set criticalMESSAGE;
 set replicaInST;
 set replicaInDYN;
 set hardMESSAGE;
 set softMESSAGE;

 set STmessage := criticalMESSAGE union hardMESSAGE union replicaInST;
```

```
 set DYNmessage := softMESSAGE union replicaInDYN;
 set MESSAGE := STmessage union DYNmessage;

 set NODE;
 set NSTM within ( NODE cross STmessage );

 set FrameID;
 set CHANNEL;
 set CYCLE;
 set SLOT;

 set CCS within (CHANNEL cross CYCLE cross SLOT);
 set CDYNM within (CHANNEL cross DYNmessage);
 set CF within (CHANNEL cross FrameID);

    check card(DYNmessage) = card(CF);

### Parameters ###

 param Msize{MESSAGE};
 param Mdeadline {STmessage union replicaInDYN};

 param Sslot;
 param STsize := card(NODE) * Sslot;
 param DYNsize;
 param Tbus := STsize + DYNsize ;

 param gdMinislot;
```

```
 param pLatestTx{m in DYNmessage} := DYNsize - Msize[m] ;

 param bignum;


### Variables ###

 var message_assign{CCS, NSTM} binary;
 var node_assign{SLOT, NSTM} binary;

 var Mcycle{STmessage};
 var Mslot{STmessage};

 var message_to_frameID{CF,CDYNM} binary;
 var Mframeid{CDYNM};

 var MM{CDYNM,CDYNM} binary;
 var TT{CDYNM,CDYNM} binary;

 var Atime{DYNmessage};
 var Btime{DYNmessage};

 var BusCycle{DYNmessage};
 var roundBusCycle{DYNmessage} integer;

 var Rtime{MESSAGE};
```

```
### Objective ###
 minimize Obj:  sum{m in MESSAGE} Rtime[m];


###------ The beginning for Constraints of ST messages ------- ###

## Message Mutual Exclusion Constraint ##

 subject to c1{(n,m) in NSTM}:
           sum {(c,i,j) in CCS} message_assign[c,i,j,n,m] = 1 ;

 subject to c2{(n,m) in NSTM}:
           sum{j in SLOT} node_assign[j,n,m] = 1;

### Message, Node and Slot Constraints ###

 subject to c3{j in SLOT,(n,p) in NSTM,(n,q) in NSTM}:
           node_assign[j,n,p] = node_assign[j,n,q];

 subject to c4{j in SLOT}:
           sum{(n,m) in NSTM} node_assign[j,n,m] >= 1;

 subject to c5{(c,i,j) in CCS}:
           sum{(n,m) in NSTM}
              message_assign[c,i,j,n,m] * Msize[m] <= Sslot ;

 subject to c6{j in SLOT, n in NODE}:
           sum{(c,i,j) in CCS, (n,m) in NSTM}
```

```
                message_assign[c,i,j,n,m]
        = sum{(n,m) in NSTM}
                node_assign[j,n,m];

### Message Deadline Constraints ###

 subject to c7{(n,m) in NSTM}:
           Mcycle[m] = sum{(c,i,j) in CCS}
                         message_assign[c,i,j,n,m] * i ;

 subject to c8{(n,m) in NSTM}:
           Mslot[m] = sum{(c,i,j) in CCS}
                        message_assign[c,i,j,n,m] * j;

 subject to c9{(n,m) in NSTM}:
           Rtime[m] = (Mcycle[m] - 1) * Tbus + Mslot[m] * Sslot;

 subject to c10{m in STmessage}:
           Rtime[m] <= Mdeadline[m];

 subject to SlotSizeConstraint:
           Sslot >= max{m in STmessage} Msize[m];

###------ The end for Constraints of ST messages ------- ###


###------ The beginning for Constraints of DYN messages ------- ###
```

```
subject to c11{(c,fid) in CF}:
           sum{(c,m) in CDYNM} message_to_frameID[c,fid,c,m] = 1;

subject to c12{(c,m) in CDYNM}:
           sum{(c,fid) in CF} message_to_frameID[c,fid,c,m] = 1;

subject to c13{(c,m) in CDYNM}:
            Mframeid[c,m] = sum{(c,fid) in CF}
                              message_to_frameID[c,fid,c,m] * fid ;

subject to c14{(c,i) in CDYNM, (c,j) in CDYNM : i<>j}:
            Mframeid[c,i] - Mframeid[c,j] <= bignum * MM[c,i,c,j] ;

subject to c15{(c,i) in CDYNM, (c,j) in CDYNM : i<>j}:
            Mframeid[c,j] - Mframeid[c,i] <= bignum * TT[c,i,c,j] ;

subject to c16{(c,i) in CDYNM, (c,j) in CDYNM : i<>j}:
            MM[c,i,c,j] + TT[c,i,c,j] = 1;

subject to c17{(c,i) in CDYNM}:
            BusCycle[i] = sum{(c,j) in CDYNM:i<>j}
                            MM[c,i,c,j] * Msize[j] / pLatestTx[i];

subject to c18{m in DYNmessage}:
            roundBusCycle[m] >= BusCycle[m];

subject to c19{(c,m) in CDYNM}:
           Atime[m] =
```

```
                Tbus - STsize - (Mframeid[c,m] - 1) * gdMinislot;

 subject to c20{m in DYNmessage}:
            Btime[m] =   roundBusCycle[m] * Tbus + STsize
                       + pLatestTx[m] * gdMinislot;

 subject to c21{m in DYNmessage}:
            Rtime[m] = Atime[m] + Btime[m] + Msize[m];

 subject to c22{m in replicaInDYN}:
            Rtime[m] <= Mdeadline[m];

###------ The end for Constraints of DYN messages ------- ###

#### The End of the Model file ####



#### The Beginning of the Data file ####

 set criticalMESSAGE :=  m1  m2  m3;
 set hardMESSAGE := m4;
 set softMESSAGE := m5  m6;

 set replicaInST := m1copy m2copy;
 set replicaInDYN := m3copy;

 set CHANNEL := A  B ;
```

```
set FrameID := 1 2;

set NODE := n1  n2  n3;

set NSTM :=
   (n1, *) m1  m1copy
   (n2, *) m2  m3  m2copy
   (n3, *) m4 ;

set CDYNM :=
   (A, *) m5  m3copy
   (B, *) m6        ;

set CF :=
   (A, *) 1 2
   (B, *) 1    ;

set CYCLE := 1 2 3 ;
set SLOT := 1 2 3;
set CCS :=
   (A, 1, *) 1 2 3
   (B, 1, *) 1 2 3
   (A, 2, *) 1 2 3
   (B, 2, *) 1 2 3
   (A, 3, *) 1 2 3
   (B, 3, *) 1 2 3 ;

param Sslot := 5;
```

```
param DYNsize := 10;
param gdMinislot := 1;
param bignum := 1000;

param:        Msize   :=
      m1       5
      m2       3
      m3       2
      m4       4
      m5       3
      m6       5
      m1copy   5
      m2copy   3
      m3copy   2;

param:        Mdeadline   :=
      m1       100
      m2       100
      m3       100
      m4       100
      m1copy   100
      m2copy   100
      m3copy   100;

#### The End of the Data file ####
```

Appendix B Data Files used for the Real-life Example

B.1 Naive Solution

```
#### The Beginning of the Data file ####

 set criticalMESSAGE := m1  m2  m3  m4  m5  m6  ;
 set hardMESSAGE := m7  m8  m9  m10  m11  m12
                    m13  m14  m15  m16  m17  m18;
 set softMESSAGE := m19  m20  m21  m22  m23  m24  m25  m26   ;

 set NODE := n1  n2  n3  n4  n5;

 set FrameID := 1 2 3 4 5 6 7 8;

 set STNM :=
```

```
   (n1, *) m13
   (n2, *) m14
   (n3, *) m7   m8
   (n4, *) m10  m15  m16  m17  m3   m6
   (n5, *) m9   m11  m12  m18  m1   m2   m4  m5 ;

set CYCLE := 1 2 3 4;

set SLOT := 1 2 3 4 5;

set CS :=
   (1, *) 1 2 3 4 5
   (2, *) 1 2 3 4 5
   (3, *) 1 2 3 4 5  ;

param Sslot := 18;
param DYNsize := 12;
param gdMinislot := 1;
param bignum := 1000;

 param:      Msize   :=
      m1       3
      m2      10
      m3       2
      m4       6
      m5       3
      m6       1
      m7      10
```

```
      m8       4
      m9       5
      m10      9
      m11      4
      m12     10
      m13      7
      m14      8
      m15      7
      m16      2
      m17      5
      m18      6
      m19      2
      m20      3
      m21      1
      m22      2
      m23      4
      m24      5
      m25      3
      m26      5    ;

param:      Mdeadline   :=
      m1       460
      m2       460
      m3       460
      m4       460
      m5       460
      m6       460
      m7       460
```

```
      m8       460
      m9       460
      m10      460
      m11      460
      m12      460
      m13      247
      m14      249
      m15      460
      m16      460
      m17      460
      m18      460    ;

#### The End of the Data file ####
```

B.2 Optimal Solution with Replicas

```
#### The Beginning of the Data file ####

 set criticalMESSAGE := m1  m2  m3  m4  m5  m6  ;
 set hardMESSAGE := m7  m8  m9  m10  m11  m12
                    m13  m14  m15  m16  m17  m18;
 set softMESSAGE := m19  m20  m21  m22  m23  m24  m25  m26   ;

 set replicaInST := m1r  m2r  m3r  m4r ;
 set replicaInDYN := m5r  m6r  ;
```

```
set CHANNEL := A  B ;

set NODE := n1  n2  n3  n4  n5;

set FrameID := 1 2 3 4 5;

set NSTM :=
   (n1, *) m13
   (n2, *) m14
   (n3, *) m7   m8
   (n4, *) m10  m15  m16  m17  m3   m6  m3r
   (n5, *) m9   m11  m12  m18  m1   m2  m4  m5   m1r  m2r  m4r ;

set CDYNM :=
   (A, *) m19  m20  m21  m22  m23
   (B, *) m24  m25  m26  m5r  m6r ;

set CF :=
   (A, *) 1 2 3 4 5
   (B, *) 1 2 3 4 5;

set CYCLE := 1 2 3;

set SLOT := 1 2 3 4 5;

set CCS :=
   (A, 1, *) 1 2 3 4 5
```

```
   (B, 1, *) 1 2 3 4 5
   (A, 2, *) 1 2 3 4 5
   (B, 2, *) 1 2 3 4 5   ;

param Sslot := 18;
param DYNsize := 12;
param gdMinislot := 1;
param bignum := 1000;

param:       Msize   :=
      m1       3
      m2      10
      m3       2
      m4       6
      m5       3
      m6       1
      m7      10
      m8       4
      m9       5
      m10      9
      m11      4
      m12     10
      m13      7
      m14      8
      m15      7
      m16      2
      m17      5
      m18      6
```

```
      m19      2
      m20      3
      m21      1
      m22      2
      m23      4
      m24      5
      m25      3
      m26      5
      m1r      3
      m2r     10
      m3r      2
      m4r      6
      m5r      3
      m6r      1     ;

param:       Mdeadline   :=
      m1       460
      m2       460
      m3       460
      m4       460
      m5       460
      m6       460
      m7       460
      m8       460
      m9       460
      m10      460
      m11      460
      m12      460
```

```
      m13      247
      m14      249
      m15      460
      m16      460
      m17      460
      m18      460
      m1r      460
      m2r      460
      m3r      460
      m4r      460
      m5r      460
      m6r      460  ;

#### The End of the Data file ####
```

Appendix C Results for the Real-life Example

C.1 Naive Solution

```
---------------------------------------------------------------------
MODEL.STATISTICS

Problem name       :naive
Pathname           :C:\Program Files\AmplStudio Modeling System 1
                   :.6.J\Bin\MyWorkSpace\
Date               :1:11:2008
Time               :09:39-09:50
Constraints        :819        :  Nonzeros
S_Constraints      :799
Variables          :638        :  Nonzeros

SOLUTION.RESULT
```

```
'Optimal solution found'

CPLEX 10.0.0: optimal integer solution; objective 3596
5478578 MIP simplex iterations
586254 branch-and-bound nodes

DECISION.VARIABLES

                Variable                Activity

    ---------------------------------------------
12    "message_assign[1,1,'n5','m11']"     1
15    "message_assign[1,1,'n5','m1']"      1
17    "message_assign[1,1,'n5','m4']"      1
18    "message_assign[1,1,'n5','m5']"      1
24    "message_assign[1,2,'n4','m15']"     1
25    "message_assign[1,2,'n4','m16']"     1
26    "message_assign[1,2,'n4','m17']"     1
27    "message_assign[1,2,'n4','m3']"      1
28    "message_assign[1,2,'n4','m6']"      1
39    "message_assign[1,3,'n3','m7']"      1
40    "message_assign[1,3,'n3','m8']"      1
55    "message_assign[1,4,'n1','m13']"     1
74    "message_assign[1,5,'n2','m14']"     1
104   "message_assign[2,1,'n5','m18']"     1
106   "message_assign[2,1,'n5','m2']"      1
113   "message_assign[2,2,'n4','m10']"     1
191   "message_assign[3,1,'n5','m9']"      1
```

```
193   "message_assign[3,1,'n5','m12']"     1
281   "node_assign[1,'n5','m9']"           1
282   "node_assign[1,'n5','m11']"          1
283   "node_assign[1,'n5','m12']"          1
284   "node_assign[1,'n5','m18']"          1
285   "node_assign[1,'n5','m1']"           1
286   "node_assign[1,'n5','m2']"           1
287   "node_assign[1,'n5','m4']"           1
288   "node_assign[1,'n5','m5']"           1
293   "node_assign[2,'n4','m10']"          1
294   "node_assign[2,'n4','m15']"          1
295   "node_assign[2,'n4','m16']"          1
296   "node_assign[2,'n4','m17']"          1
297   "node_assign[2,'n4','m3']"           1
298   "node_assign[2,'n4','m6']"           1
309   "node_assign[3,'n3','m7']"           1
310   "node_assign[3,'n3','m8']"           1
325   "node_assign[4,'n1','m13']"          1
344   "node_assign[5,'n2','m14']"          1
361   "Mcycle['m1']"                       1
362   "Mcycle['m2']"                       2
363   "Mcycle['m3']"                       1
364   "Mcycle['m4']"                       1
365   "Mcycle['m5']"                       1
366   "Mcycle['m6']"                       1
367   "Mcycle['m7']"                       1
368   "Mcycle['m8']"                       1
369   "Mcycle['m9']"                       3
```

```
370   "Mcycle['m10']"                    2
371   "Mcycle['m11']"                    1
372   "Mcycle['m12']"                    3
373   "Mcycle['m13']"                    1
374   "Mcycle['m14']"                    1
375   "Mcycle['m15']"                    1
376   "Mcycle['m16']"                    1
377   "Mcycle['m17']"                    1
378   "Mcycle['m18']"                    2
379   "Mslot['m1']"                      1
380   "Mslot['m2']"                      1
381   "Mslot['m3']"                      2
382   "Mslot['m4']"                      1
383   "Mslot['m5']"                      1
384   "Mslot['m6']"                      2
385   "Mslot['m7']"                      3
386   "Mslot['m8']"                      3
387   "Mslot['m9']"                      1
388   "Mslot['m10']"                     2
389   "Mslot['m11']"                     1
390   "Mslot['m12']"                     1
391   "Mslot['m13']"                     4
392   "Mslot['m14']"                     5
393   "Mslot['m15']"                     2
394   "Mslot['m16']"                     2
395   "Mslot['m17']"                     2
396   "Mslot['m18']"                     1
397   "Rtime['m1']"                     18
```

```
398   "Rtime['m2']"                      120
399   "Rtime['m3']"                       36
400   "Rtime['m4']"                       18
401   "Rtime['m5']"                       18
402   "Rtime['m6']"                       36
403   "Rtime['m7']"                       54
404   "Rtime['m8']"                       54
405   "Rtime['m9']"                      222
406   "Rtime['m10']"                     138
407   "Rtime['m11']"                      18
408   "Rtime['m12']"                     222
409   "Rtime['m13']"                      72
410   "Rtime['m14']"                      90
411   "Rtime['m15']"                      36
412   "Rtime['m16']"                      36
413   "Rtime['m17']"                      36
414   "Rtime['m18']"                     120
415   "Rtime['m19']"                     214
416   "Rtime['m20']"                     114
417   "Rtime['m21']"                     212
418   "Rtime['m22']"                     213
419   "Rtime['m23']"                     312
420   "Rtime['m24']"                     413
421   "Rtime['m25']"                     215
422   "Rtime['m26']"                     313
424   "message_to_frameID[1,'m20']"        1
437   "message_to_frameID[2,'m25']"        1
439   "message_to_frameID[3,'m19']"        1
```

```
450   "message_to_frameID[4,'m22']"        1
457   "message_to_frameID[5,'m21']"        1
470   "message_to_frameID[6,'m26']"        1
475   "message_to_frameID[7,'m23']"        1
484   "message_to_frameID[8,'m24']"        1
487   "Mframeid['m19']"                    3
488   "Mframeid['m20']"                    1
489   "Mframeid['m21']"                    5
490   "Mframeid['m22']"                    4
491   "Mframeid['m23']"                    7
492   "Mframeid['m24']"                    8
493   "Mframeid['m25']"                    2
494   "Mframeid['m26']"                    6
607   "Atime['m19']"                      10
608   "Atime['m20']"                      12
609   "Atime['m21']"                       8
610   "Atime['m22']"                       9
611   "Atime['m23']"                       6
612   "Atime['m24']"                       5
613   "Atime['m25']"                      11
614   "Atime['m26']"                       7
615   "Btime['m19']"                     202
616   "Btime['m20']"                      99
617   "Btime['m21']"                     203
618   "Btime['m22']"                     202
619   "Btime['m23']"                     302
620   "Btime['m24']"                     403
621   "Btime['m25']"                     201
```

```
622   "Btime['m26']"                    301
623   "BusCycle['m19']"                   0.6
625   "BusCycle['m21']"                   0.909091
626   "BusCycle['m22']"                   0.8
627   "BusCycle['m23']"                   2
628   "BusCycle['m24']"                   2.85714
629   "BusCycle['m25']"                   0.333333
630   "BusCycle['m26']"                   1.57143
631   "roundBusCycle['m19']"              1
633   "roundBusCycle['m21']"              1
634   "roundBusCycle['m22']"              1
635   "roundBusCycle['m23']"              2
636   "roundBusCycle['m24']"              3
637   "roundBusCycle['m25']"              1
638   "roundBusCycle['m26']"              2

-----------------------------------------------------
```

C.2 Optimal Solution with Replicas

```
-------------------------------------------------------------------
MODEL.STATISTICS


Problem name        :optimal
Pathname            :C:\Program Files\AmplStudio Modeling System 1
                    :.6.J\Bin\MyWorkSpace\
Date                :1:6:2008
```

```
Time                :16:12---16:15
Constraints         :1275       :  Nonzeros
S_Constraints       :1140
Variables           :1196       :  Nonzeros

SOLUTION.RESULT

'Optimal solution found'

CPLEX 10.0.0: optimal integer solution; objective 3166
1318528 MIP simplex iterations
237849 branch-and-bound nodes

DECISION.VARIABLES

                    Variable                    Activity

    ------------------------------------------------------
12     "message_assign['A',1,1,'n5','m9']"       1
16     "message_assign['A',1,1,'n5','m1']"       1
17     "message_assign['A',1,1,'n5','m2']"       1
27     "message_assign['A',1,2,'n4','m10']"      1
30     "message_assign['A',1,2,'n4','m17']"      1
48     "message_assign['A',1,3,'n3','m8']"       1
90     "message_assign['A',1,5,'n2','m14']"      1
123    "message_assign['B',1,1,'n5','m11']"      1
128    "message_assign['B',1,1,'n5','m4']"       1
129    "message_assign['B',1,1,'n5','m5']"       1
130    "message_assign['B',1,1,'n5','m1r']"      1
```

```
138    "message_assign['B',1,2,'n4','m15']"      1
139    "message_assign['B',1,2,'n4','m16']"      1
141    "message_assign['B',1,2,'n4','m3']"       1
142    "message_assign['B',1,2,'n4','m6']"       1
143    "message_assign['B',1,2,'n4','m3r']"      1
157    "message_assign['B',1,3,'n3','m7']"       1
177    "message_assign['B',1,4,'n1','m13']"      1
234    "message_assign['A',2,1,'n5','m12']"      1
242    "message_assign['A',2,1,'n5','m4r']"      1
345    "message_assign['B',2,1,'n5','m18']"      1
351    "message_assign['B',2,1,'n5','m2r']"      1
672    "node_assign[1,'n5','m9']"                1
673    "node_assign[1,'n5','m11']"               1
674    "node_assign[1,'n5','m12']"               1
675    "node_assign[1,'n5','m18']"               1
676    "node_assign[1,'n5','m1']"                1
677    "node_assign[1,'n5','m2']"                1
678    "node_assign[1,'n5','m4']"                1
679    "node_assign[1,'n5','m5']"                1
680    "node_assign[1,'n5','m1r']"               1
681    "node_assign[1,'n5','m2r']"               1
682    "node_assign[1,'n5','m4r']"               1
687    "node_assign[2,'n4','m10']"               1
688    "node_assign[2,'n4','m15']"               1
689    "node_assign[2,'n4','m16']"               1
690    "node_assign[2,'n4','m17']"               1
691    "node_assign[2,'n4','m3']"                1
692    "node_assign[2,'n4','m6']"                1
```

```
693    "node_assign[2,'n4','m3r']"          1
707    "node_assign[3,'n3','m7']"           1
708    "node_assign[3,'n3','m8']"           1
727    "node_assign[4,'n1','m13']"          1
750    "node_assign[5,'n2','m14']"          1
771    "Mcycle['m1']"                       1
772    "Mcycle['m2']"                       1
773    "Mcycle['m3']"                       1
774    "Mcycle['m4']"                       1
775    "Mcycle['m5']"                       1
776    "Mcycle['m6']"                       1
777    "Mcycle['m7']"                       1
778    "Mcycle['m8']"                       1
779    "Mcycle['m9']"                       1
780    "Mcycle['m10']"                      1
781    "Mcycle['m11']"                      1
782    "Mcycle['m12']"                      2
783    "Mcycle['m13']"                      1
784    "Mcycle['m14']"                      1
785    "Mcycle['m15']"                      1
786    "Mcycle['m16']"                      1
787    "Mcycle['m17']"                      1
788    "Mcycle['m18']"                      2
789    "Mcycle['m1r']"                      1
790    "Mcycle['m2r']"                      2
791    "Mcycle['m3r']"                      1
792    "Mcycle['m4r']"                      2
793    "Mslot['m1']"                        1
```

```
794    "Mslot['m2']"                              1
795    "Mslot['m3']"                              2
796    "Mslot['m4']"                              1
797    "Mslot['m5']"                              1
798    "Mslot['m6']"                              2
799    "Mslot['m7']"                              3
800    "Mslot['m8']"                              3
801    "Mslot['m9']"                              1
802    "Mslot['m10']"                             2
803    "Mslot['m11']"                             1
804    "Mslot['m12']"                             1
805    "Mslot['m13']"                             4
806    "Mslot['m14']"                             5
807    "Mslot['m15']"                             2
808    "Mslot['m16']"                             2
809    "Mslot['m17']"                             2
810    "Mslot['m18']"                             1
811    "Mslot['m1r']"                             1
812    "Mslot['m2r']"                             1
813    "Mslot['m3r']"                             2
814    "Mslot['m4r']"                             1
819    "message_to_frameID['A',1,'A','m23']"      1
826    "message_to_frameID['A',2,'A','m20']"      1
835    "message_to_frameID['A',3,'A','m19']"      1
848    "message_to_frameID['A',4,'A','m22']"      1
857    "message_to_frameID['A',5,'A','m21']"      1
874    "message_to_frameID['B',1,'B','m6r']"      1
881    "message_to_frameID['B',2,'B','m25']"      1
```

```
890    "message_to_frameID['B',3,'B','m24']"     1
903    "message_to_frameID['B',4,'B','m5r']"     1
912    "message_to_frameID['B',5,'B','m26']"     1
915    "Mframeid['A','m19']"                     3
916    "Mframeid['A','m20']"                     2
917    "Mframeid['A','m21']"                     5
918    "Mframeid['A','m22']"                     4
919    "Mframeid['A','m23']"                     1
920    "Mframeid['B','m24']"                     3
921    "Mframeid['B','m25']"                     2
922    "Mframeid['B','m26']"                     5
923    "Mframeid['B','m5r']"                     4
924    "Mframeid['B','m6r']"                     1
1125   "Atime['m19']"                           10
1126   "Atime['m20']"                           11
1127   "Atime['m21']"                            8
1128   "Atime['m22']"                            9
1129   "Atime['m23']"                           12
1130   "Atime['m24']"                           10
1131   "Atime['m25']"                           11
1132   "Atime['m26']"                            8
1133   "Atime['m5r']"                            9
1134   "Atime['m6r']"                           12
1135   "Btime['m19']"                          202
1136   "Btime['m20']"                          201
1137   "Btime['m21']"                          203
1138   "Btime['m22']"                          202
1139   "Btime['m23']"                           98
```

```
1140   "Btime['m24']"                    199
1141   "Btime['m25']"                    201
1142   "Btime['m26']"                    301
1143   "Btime['m5r']"                    201
1144   "Btime['m6r']"                    101
1145   "BusCycle['m19']"                   0.7
1146   "BusCycle['m20']"                   0.444444
1147   "BusCycle['m21']"                   1
1148   "BusCycle['m22']"                   0.9
1150   "BusCycle['m24']"                   0.571429
1151   "BusCycle['m25']"                   0.111111
1152   "BusCycle['m26']"                   1.71429
1153   "BusCycle['m5r']"                   1
1155   "roundBusCycle['m19']"              1
1156   "roundBusCycle['m20']"              1
1157   "roundBusCycle['m21']"              1
1158   "roundBusCycle['m22']"              1
1160   "roundBusCycle['m24']"              1
1161   "roundBusCycle['m25']"              1
1162   "roundBusCycle['m26']"              2
1163   "roundBusCycle['m5r']"              1
1165   "Rtime['m1']"                     18
1166   "Rtime['m2']"                     18
1167   "Rtime['m3']"                     36
1168   "Rtime['m4']"                     18
1169   "Rtime['m5']"                     18
1170   "Rtime['m6']"                     36
1171   "Rtime['m7']"                     54
```

```
1172   "Rtime['m8']"                          54
1173   "Rtime['m9']"                          18
1174   "Rtime['m10']"                         36
1175   "Rtime['m11']"                         18
1176   "Rtime['m12']"                        120
1177   "Rtime['m13']"                         72
1178   "Rtime['m14']"                         90
1179   "Rtime['m15']"                         36
1180   "Rtime['m16']"                         36
1181   "Rtime['m17']"                         36
1182   "Rtime['m18']"                        120
1183   "Rtime['m1r']"                         18
1184   "Rtime['m2r']"                        120
1185   "Rtime['m3r']"                         36
1186   "Rtime['m4r']"                        120
1187   "Rtime['m19']"                        214
1188   "Rtime['m20']"                        215
1189   "Rtime['m21']"                        212
1190   "Rtime['m22']"                        213
1191   "Rtime['m23']"                        114
1192   "Rtime['m24']"                        214
1193   "Rtime['m25']"                        215
1194   "Rtime['m26']"                        314
1195   "Rtime['m5r']"                        213
1196   "Rtime['m6r']"                        114

-----------------------------------------------------
```