DEVELOPMENT OF STRATEGIES FOR REDUCING THE WORST-CASE MESSAGE RESPONSE TIMES ON THE CONTROLLER AREA NETWORK

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

VAKKAS ÇELIK

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

JANUARY 2012

Approval of the thesis:

DEVELOPMENT OF STRATEGIES FOR REDUCING THE WORST-CASE MESSAGE RESPONSE TIMES ON THE CONTROLLER AREA NETWORK

submitted by **VAKKAS ÇELIK** in partial fulfillment of the requirements for the degree of **Master of Science in Electrical and Electronics Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. İsmet Erkmen
Head of Department, **Electrical and Electronics Engineering** _____

Assoc. Prof. Dr. Şenan Ece GÜRAN SCHMIDT
Supervisor, **Electrical-Electronics Engineering Dept., METU** _____

Assist. Prof. Dr. Klaus Schmidt
Co-supervisor, **Mechatronics Engineering Dept., ÇANKAYA UNIVERSITY** _____

Examining Committee Members:

Prof. Dr. Semih BİLGEN
Electrical-Electronics Engineering Dept., METU _____

Assoc. Prof. Dr. Şenan Ece GÜRAN SCHMIDT
Electrical-Electronics Engineering Dept., METU _____

Prof. Dr. Kemal LEBLEBİCİOĞLU
Electrical-Electronics Engineering Dept., METU _____

Assoc. Prof. Dr. Cüneyt BAZLAMAÇCI
Electrical-Electronics Engineering Dept., METU _____

Prof. Dr. Celal Zaim ÇİL
Electronic and Communication Engineering Dept., Çankaya University _____

Date: _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: VAKKAS ÇELIK

Signature :

ABSTRACT

DEVELOPMENT OF STRATEGIES FOR REDUCING THE WORST-CASE MESSAGE RESPONSE TIMES ON THE CONTROLLER AREA NETWORK

Çelik, Vakkas

M.S., Department of Electrical and Electronics Engineering

Supervisor : Assoc. Prof. Dr. Şenan Ece GÜRAN SCHMIDT

Co-Supervisor : Assist. Prof. Dr. Klaus Schmidt

JANUARY 2012, 62 pages

The controller area network (CAN) is the de-facto standard for in-vehicle communication. The growth of time-critical applications in modern cars leads to a considerable increase in the message traffic on CAN. Hence, it is essential to determine efficient message schedules on CAN that guarantee that all communicated messages meet their timing constraints. The aim of this thesis is to develop offset scheduling strategies that find feasible schedules for higher bus load levels compared to conventional CAN scheduling approaches. We formulate the offset scheduling as a constraint optimization problem that maximizes the sum of message slacks where slack is defined as the difference between the deadline and the worst-case response time (WCRT) of a message. The constraint to ensure the feasibility of the schedules is keeping all slacks positive. In this respect we propose two heuristic offset scheduling algorithms which integrate an existing method for the WCRT analysis in the schedule computation. We apply our algorithms to various examples and compare the results with a well-known offset scheduling algorithm. The results show that our algorithms can generate feasible schedules at significantly high loads with run times shorter than 5 minutes.

Keywords: CAN, Offset Scheduling, WCRT Analysis, Genetic Algorithm

ÖZ

DENETLEYİCİ ALAN AĞI(CAN) ÜZERİNDEKİ EN KÖTÜ DURUMDAKİ MESAJ TEPKİ SÜRELERİNİ AZALTMAK İÇİN STRATEJİLER GELİŞTİRME

Çelik, Vakkas
Yüksek Lisans, Elektrik-Elektronik Mühendisliği Bölümü
Tez Yöneticisi : Doç. Dr. Şenan Ece Güran Schmidt
Ortak Tez Yöneticisi : Y. Doç. Dr. Klaus Schmidt

Ocak 2012, 62 sayfa

Denetleyici Alan Ağı (CAN), araç içi iletişimde bilfiil kullanılan standarttır. Modern arabalardaki zaman-kritik uygulamaların artması CAN hattındaki mesaj trafiğinin artmasına neden olmuştur. Bu yüzden, gönderilen mesajların adreslerine zamanlarında ulaştığını garanti etmek için CAN trafiğinde mesaj çizelgelerinin verimli şekilde yapılması gerekmektedir. Bu tezin amacı, bilinen CAN çizelgeleme yaklaşımlarına kıyasla daha fazla veriyolu yükü altında uygulanabilir gecikme çizelgeleri geliştirmektir. Buna göre gecikme çizelgelemesi, bir mesajın son varış süresi ile en kötü durumdaki tepki süresi arasındaki fark olarak tanımlanan gevşekliği maksimuma çıkarmaya çalışan bir kısıtlı optimizasyon problemi olarak formüle edilmektedir. Çizelgelerin uygulanabilirliğini sağlayan bu kısıtlama, tüm gevşeklikleri pozitif tutmaktır. Bu bakımdan, gecikme çizelgesi hesaplamasında halihazırdaki gecikme çizelgeleme yöntemini mesajların en kötü durumdaki tepki sürelerinin analiziyle bütünleştiren iki adet sezgisel gecikme çizelgeleme algoritması önerilmektedir. Bu algoritmalar farklı örneklerle denenerek, iyi bilinen gecikme çizelgeleme algoritması ile kıyaslanmaktadır. Elde edilen sonuçlar tezde geliştirilen algoritmaların yoğun trafiklerde bile 5 dakikadan az sürede uygulanabilir mesaj çizelgeleri ürettiğini göstermektedir.

Anahtar Kelimeler: Denetleyici Alan Ağı, Gecikme Çizelgeleme, Genetik Algoritma

To My Wife

ACKNOWLEDGMENTS

First of all, I would like to thank my thesis supervisor Assoc. Prof. Dr. Şenan Ece Schmidt and my thesis co-supervisor Assist. Prof. Dr. Klaus Schmidt for their valuable critics, excellent supervision and support. Their help, guidance, suggestions and encouragement have been very important for me throughout this period.

This thesis would not have been possible without my wife's encouragement, support and love. I would like to thank her for her personal support and great patience at all times. I would like to send my special thanks to my mom. Eventhough she has been far away from me, I have felt her support and prayings during this thesis period.

I wish to thank TUBITAK UZAY for giving me the opportunity of continuing my education. It gives me great pleasure in acknowledging the support and help of my ex-supervisor Orhan ŞENGÜL and new supervisors Mustafa Sancay KIRIK and Onur HALİLOĞLU. I am indebted to my colleagues for their close friendship. I would like to thank them seperately. Fethi, Erol, Emre, Yakup Murat, Özgür S., Özgür Y., Ömer, Levent, Aziz, Derya, Himmet, Sinan and Muhsin bring joy and happiness to me always.

TABLE OF CONTENTS

ABSTRACT iv

ÖZ v

ACKNOWLEDGMENTS viii

TABLE OF CONTENTS ix

LIST OF TABLES xi

LIST OF FIGURES xii

LIST OF ABBREVIATIONS AND ACRONYMS xiv

CHAPTERS

1 INTRODUCTION 1

2 CAN Protocol 5

2.1 Description of the CAN Protocol 5

2.2 Schedulability Analysis and Generalized CAN Scheduling Algorithm 6

2.2.1 Scheduling Model Used in This Thesis 6

2.2.2 Classical CAN Schedulability Analysis 8

2.3 Generalized CAN Scheduling Algorithm 11

3 CAN Scheduling with Offsets 13

3.1 Offset Scheduling Description 13

3.2 Standard Offset Scheduling Algorithm (SOSA) 14

3.2.1 Design Hypothesis and Notations 15

3.2.2 Description of the Algorithm 15

3.2.3 The Algorithm 16

3.2.4 Application of the Algorithm 16

3.3 Schedulability Analysis for Offset Scheduling 18

3.3.1 Description of the Algorithm 18
3.3.2 The Algorithm 27
4 Developed Algorithms for Offset Scheduling 28
4.1 Problems and Challenges 28
4.1.1 Problem 28
4.1.2 Challenges 28
4.1.3 Offset Scheduling as Optimization Problem 30
4.2 Local Neighborhood Search Algorithm (LNSA) 31
4.2.1 Description of LNSA 31
4.2.2 Illustration of LNSA 32
4.3 Genetic Algorithm Approach 34
4.3.1 General Description 34
4.3.2 Genetic Algorithm for Offset Scheduling (GAOS) 35
5 Experimental Evaluation of the Developed Offset Scheduling Algorithms . . 39
5.1 Development Environment 39
5.2 Existing Functionality of the Automotive Scheduling Software . . . 39
5.3 Implemented Functions in the Scope of the Thesis 40
5.4 Description of NETCARBENCH 41
5.5 Generated Message Sets 41
5.6 Comparison and Discussion 42
5.6.1 Test Results Obtained for Class A 42
5.6.2 Test Results Obtained for Class B 45
5.6.3 Test Results Obtained for Class C 48
5.6.4 Test Results Obtained for Class D 51
5.6.5 Test Results Obtained for Class E 53
5.6.6 Test Results Obtained for GAOS with the Solution of LNSA 55
5.6.7 General Observation on the Results 57
6 CONCLUSION 59
REFERENCES 61

LIST OF TABLES

TABLES

Table 2.1 Key Notations in This Thesis 8

Table 3.1 Example CAN Network Messages and Their Properties 17

Table 3.2 The release array A of node N_1 17

Table 3.3 Example CAN Network Messages and Their Properties 18

Table 3.4 Iteration 1 24

Table 3.5 Iteration 2 25

Table 3.6 Iteration 3 25

Table 3.7 Iteration 4 26

Table 3.8 Iteration 5 26

Table 5.1 Configuration of the Message Sets Used in the Experiments 42

LIST OF FIGURES

FIGURES

Figure 2.1 Standard CAN Frame 6

Figure 3.1 An example CAN Network with Offset Assigned Messages 14

Figure 3.2 Illustration of hyper-period and feasible interval 20

Figure 3.3 Possible Synchronized Messages of message m_8 23

Figure 4.1 Example initial population to use in Genetic Algorithm 36

Figure 4.2 Illustration of Crossover Used for Generating New Schedules. (a) and (b) are the parent schedules, and (c), (d) and (e) are the child schedules generated from (a) and (b) 36

Figure 4.3 Illustration of mutation applied to child schedules. (a), (b) and (c) are the child schedules which mutation is applied with 1/7 mutation probability. 37

Figure 5.1 Schedulability Performance of the Algorithms for Class A 43

Figure 5.2 Average Number of Unscheduled Messages for Class A 44

Figure 5.3 Average Slacks for Class A 44

Figure 5.4 Average Run-Time for Class A 45

Figure 5.5 Schedulability Performance of the Algorithms for Class B 46

Figure 5.6 Average Number of Unscheduled Messages for Class B 46

Figure 5.7 Average Slacks for Class B 47

Figure 5.8 Average Run-Time for Class B 47

Figure 5.9 Schedulability Performance of the Algorithms for Class C 48

Figure 5.10 Average Number of Unscheduled Messages for Class C 49

Figure 5.11 Average Slacks for Class C 49

Figure 5.12 Average Run-Time for Class C 50
Figure 5.13 Schedulability Performance of the Algorithms for Class D 51
Figure 5.14 Average Number of Unscheduled Messages for Class D 51
Figure 5.15 Average Slacks for Class D 52
Figure 5.16 Average Run-Time for Class D 52
Figure 5.17 Schedulability Performance of the Algorithms for Class E 53
Figure 5.18 Average Number of Unscheduled Messages for Class E 54
Figure 5.19 Average Slacks for Class E 54
Figure 5.20 Average Run-Time for Class E 55
Figure 5.21 Schedulability Performance of the Algorithms for Class E 56
Figure 5.22 Average Run-Time for Class E 56

LIST OF ABBREVIATIONS AND ACRONYMS

CAN	Controller Area Network
ECU	Electronic Control Unit
WCRT	Worst Case Response Time
SOSA	Standard Offset Scheduling Algorithm
LNSA	Local Neighborhood Search Algorithm
GAOS	Genetic Algorithm for Offset Scheduling
API	Application Programming Interface
CSMA/CR	Carrier Sense Multiple Access/Collision Resolution

CHAPTER 1

INTRODUCTION

In contemporary vehicles a large number of electronic control units (ECU) exchange information enclosed in *messages*, that are transmitted over a communication bus to realize the controlling of the vehicle as well as supporting its safety. These messages can be periodic as well as sporadic. The *periodic messages* contain regularly sampled data from the sensors and information for their respective control actions while the *sporadic messages* are associated with events triggered by the driving environment and the driver. Currently the most widely used bus standard for in-vehicle communication is Controller Area Network (CAN) bus [1]. CAN bus provides event-triggered communication with a non-preemptive priority based arbitration among the messages released by the ECUs.

As the technology improves, the number of ECUs communicating on CAN bus and the amount of data exchanged increase. This yields an increase of the CAN bus load; which delays the messages including the time-critical ones. A car manufacturer has to make sure that all messages in the network are *schedulable*; i.e. worst case response times of the messages are shorter than a pre-specified *deadline* to ensure that the freshness of the data is still acceptable at the receiver side. In this case, the schedule is denoted as *feasible*. The worst-case response times of the messages increase as the load on CAN bus increases, hence, the bus utilization is supposed to be kept at low levels (up to %40) for contemporary vehicles [3]. It has to be noted that new technologies, such as FlexRay [2], bring higher bandwidth capability than CAN. However, CAN is still the most cost effective protocol and will most likely be used for at least two more decades in automotive industry [3].

The design for the in-vehicle communication includes constructing the *message set* which describes the period, length and the deadlines of the messages to be exchanged. In this respect,

the period for the sporadic messages indicates the smallest time between two consecutive message releases. Then, the message priorities which are indicated by the CAN IDs are assigned. With the additional information about the release times of the messages, it is possible to check if the messages will meet their deadlines for a given message set and the chosen CAN ID assignment [4].

Considering a CAN message set, the length, the period and the deadline of each message are decided according to the requirements of the application running on the ECUs. Then, the *schedulability* of these messages depends on the assignment of the CAN IDs and the release times of the messages from the ECUs. In principle, it is possible to algorithmically determine a feasible schedule by applying the non-preemptive task scheduling method developed in [5]. Specializations of this algorithm for CAN networks are presented in [4, 6, 7]. However, it has to be noted, that CAN IDs usually cannot be freely assigned in practical applications, since the vehicle manufacturers maintain their messages and CAN ID assignments for compatibility. Hence, it is a worthwhile task to find methods that enable the computation of feasible CAN schedules without modifying the CAN IDs.

Such method was first introduced by Grenier, Havet and Navet [8] by observing that the release times of messages impact on their worst-case response times. Here, the delay of the release of the first instance of a message is called *offset*. Introducing such offsets in the release times allows to reduce message response times, since worst-case scenarios, where many messages are released at the same time, are avoided. To this end, for a given CAN ID assignment, *offset scheduling* decides the offset for each message so as to achieve a feasible schedule.

Although a scheduling algorithm runs offline, it has to terminate in a bounded amount of time. Hence, a brute-force approach to find the offsets for CAN scheduling is not possible, since the search space is too large to be fully explored. In particular, the runtime grows exponentially with the number of messages as well as the periods of the messages. In order to overcome this problem, Grenier et. al. proposed a heuristic solution that has low-complexity [3, 8]. We call this algorithm *Standard Offset Scheduling Algorithm (SOSA)* in this thesis. The main purpose of this algorithm is to distribute the message release times of individual CAN nodes (ECUs) as uniformly as possible over time, in order to avoid synchronous message releases, since such synchronous releases lead to traffic peaks and cause large message response times. Although

the algorithm allows finding offset schedules for moderate traffic loads of about 50%, feasible schedules for higher loads cannot be achieved. As is shown in this thesis, the main reason for this deficiency is neglecting the dependencies among messages that are released from different nodes. The algorithm in [3, 8] assigns the offsets for the messages of each node independently and does not include any WCRT analysis in the schedule computation.

Objective of the thesis is to improve offset scheduling in order to find feasible offset schedules for larger bus loads. In order to achieve this task, the WCRT analysis for offset schedules proposed in [9] is used to get feedback about the offset schedule and so integrated in the schedule computation. It is observed that the *slacks* of all messages have to be positive in order to find a feasible schedule, where slack is defined as the difference between the deadline and the WCRT of message. First contribution of this thesis is to reformulate the offset scheduling as a constraint optimization problem that maximizes the sum of message slacks, while ensuring that all slacks remain positive. As the second contribution, two heuristic algorithms are proposed for the solution of this optimization problem. First one is called *Local Neighborhood Search Algorithm (LNSA)* in this thesis. *LNSA* starts from *Standard Offset Schedule* and proceeds by changing a single offset at each iteration while always keeping the best solution. *LNSA* visits each message in the system one time, and tries all possible offsets of the message one-by-one while keeping offsets of other messages the same. Second proposed algorithm is an application of *Genetic Algorithm for Offset Scheduling*. We call this algorithm *GAOS* in short. *GAOS* starts from generating an initial population based on the Standard Offset Schedule and uses genetic operators such as mutation and crossover to generate new schedules in order to reach a feasible solution by using the WCRT analysis for offset schedules to evaluate the fitness of the schedules.

In this thesis, all algorithms are implemented as an extension of an existing C++-library for automotive scheduling [10]. In order to evaluate the performances of the scheduling algorithms, a large number of message sets are used for different bus loads. These messages are generated by NETCARBENCH which allows users to define the parameters of the generated message sets [11]. Our experimental results verify that *SOSA* can easily find feasible schedules for bus loads up to 50% as claimed in [3]. It is further shown that the developed algorithms in this thesis significantly improve the offset schedule computation. They find a schedulable solution for most of the message sets at bus loads up to 80%. It is deduced from the results that GAOS is better suited for smaller bus load sets, while LNSA is better suited

for larger bus loads because of run time. It is also indicated that concatenation of LNSA and GAOS finds feasible schedules in 90% of the test cases for bus loads up to 80%. It has to be noted that run times of the developed algorithms are obviously higher than run times of *SOSA*; however, they are bounded by 5 minutes in our experiments which are conducted by a standard personal computer. Hence, the developed algorithms in this thesis are well-suited for practical applications.

The rest of the thesis is organized as follows. In Chapter 2, fundamental features of CAN protocol are described. The scheduling model and notations that are used in the thesis are defined. Moreover, the principles of generalized CAN scheduling algorithm and its schedulability analysis is explained. Detailed description of the offset scheduling concept is provided in Chapter 3. Furthermore, the first offset scheduling algorithm in the literature is discussed. This chapter ends with the analysis methodology of the worst-case response time for the offset scheduling. Chapter 4, introduces the problems and challenges to be solved in this thesis. Performance metrics used in comparing different solutions are defined. Then, our two different proposed solutions which are *GAOS* and *LNSA* are presented. Chapter 5 is devoted to the implementation of the scheduling algorithms and their evaluations. Finally, Chapter 6, includes the discussions and concluding remarks.

CHAPTER 2

CAN Protocol

2.1 Description of the CAN Protocol

CAN is an asynchronous multi-master serial vehicle bus that connects devices, sensors and actuators in a system or sub-system for real-time control applications. It was created by Robert Bosch GmbH in 1983 to provide a cost-effective communications bus for in-car electronics. The automotive industry quickly adopted CAN and, in 1993, it became the international standard known as ISO 11898. Currently, CAN is de-facto standard for in-vehicle data transmision. It is ensured that CAN will be used for many years because the multibillion dollar infrastructure that exists to support CAN provides low-cost components and a large technical support base [12].

The CAN communication protocol is a Carrier Sense Multiple Access/Collision Resolution (CSMA/CR) protocol. It is required by the CAN protocol that nodes do not attemp transmiting when bus is not idle. CAN is capable of operating at data rates of up to 1 Mbit/s on twisted pair of copper wires.

The standard CAN 2.0A data frame contains *start of frame (SOF)* bit, 12 bits of *arbitration field* with 11-bit ID, 6 bits of *control field*,up to 8 B of data, 15 bits of *cyclic redundancy check (CRC) field*, 3 bits of *acknowledgement slot (ACK)* and 7 bits of *end of frame (EOF) field*. Together, the *frame duration t* for *b* data bytes on network having *D* bit rate is [4]

$$t = \frac{55 + 10b}{D} \quad (for\ 11 - \text{bit}\ ID) \quad \text{and} \quad t = \frac{80 + 10b}{D} \quad (for\ 29 - \text{bit}\ ID).$$

In CAN network, each node is able to access to the bus and may begin to transmit if the bus

is free. This conflict is handled by a non-destructive arbitration process. When more than one node attempts to transmit a bit on the CAN bus at the same time, the bus transmits the result of the logical AND of these bits. When a node observes same polarity as it sends, then it goes on transmission; otherwise it stops transmission and starts waiting for the bus to be idle. This behaviour is used to resolve collisions. The standard CAN frame contains identifier bits of length 11 or 29 bits at the beginning of CAN frame and each CAN frame has an unique CAN ID which determines the priority of the message when accessing the network. Lower CAN IDs have higher priority and a CAN message that is transmitted with highest priority wins the arbitration.

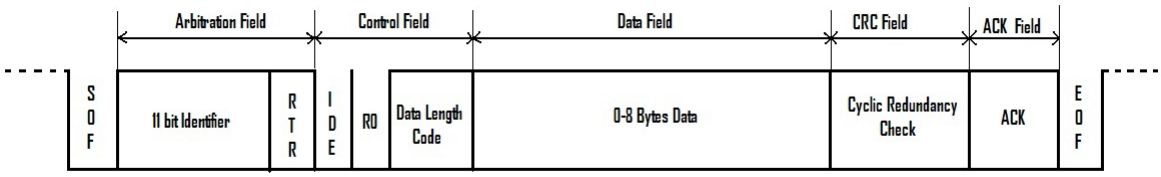

Figure 2.1: Standard CAN Frame

2.2 Schedulability Analysis and Generalized CAN Scheduling Algorithm

2.2.1 Scheduling Model Used in This Thesis

In this thesis, notations in Table 2.1 are used in order to be consistent in the whole document. A CAN system is composed of a number of nodes connected via CAN bus and this number is denoted by N_{max}. Each node in the system ensures that highest priority message queued at that node wins the arbitration at any time when arbitration starts.

Node where message m is assigned to is symbolized by N_m and each message is only generated by a unique node. Mapping of message m to priority level-m is indicated by V_m. Each message m carries between 0-8 data bytes and number of data bytes carried by message m is denoted by G_m.

The system contains a static set of hard real-time messages and this set is denoted by Z. Each message in the network is statically assigned to a node on the network. M is the number of messages in Z. Each message m has a fixed identifier, hence an unique priority. The longest time taken to transmit a given message m is denoted as C_m.

Each task has a minimum inter-arrival time termed the *period*. Note that the period is a minimum time between subsequent arrivals, rather than a strict fixed interval. If the message queued by a given task is potentially sent each time the task is invoked, then the message inherits a period equal to the period of the task. We denote as T_m, the period of a given message m.

Each message has a hard *deadline* D_m, corresponding to the maximum permitted time from occurrence of the initiating event to the end of successful transmission of the message, at which time the message data is assumed to be available on the receiving nodes that require it.

When a CAN message is generated, it can be transmitted directly if any CAN message is not already being transmitted or there are not higher priority messages in the queue waiting to be transmitted. In other case, it may wait the transmissions of higher priority messages in the queue and already being transmitted message. It is important to know how much time might be needed in the worst case in order to guarantee that the message will be transmitted on time. The worst-case response time R_m, of a message is defined as the longest time from the initiating event occurring to the message being received by the nodes that require it.

In the thesis, *absolute slack time* of message m is defined as the difference between the deadline (D_m) and the worst-case response time (R_m) and denoted by $S\,t_m$; $S\,t_m = D_m - R_m$. It captures the margin of each messages that is left until the deadline. In this thesis we define the *relative slack* of message as the ratio of slack time of a message to its deadline and notation of slack of message m is S_m. The relative slack is introduced in order to make the slacks of messages with different deadlines comparable.

$$S_m = S\,t_m / D_m = (D_m - R_m) / D_m \tag{2.1}$$

A message is *schedulable* if $R_m \leq D_m$ or, equivalently, $S_m \geq 0$. The system is schedulable iff all of the messages in the system are schedulable. Average slack of a system is the average of the relative slacks of all messages in that system and is denoted by S_{avg}.

$$S_{avg} = \left(\sum_{1 \leq i \leq M} S_i \right) / M \tag{2.2}$$

In Section 4, we will try to maximize the average slack as a measure of the quality of a

schedule.

Table 2.1: Key Notations in This Thesis

m	message m and its priority
N_{max}	number of nodes in the giving network
Z	message set in the network
M	size of Z
N_m	node message m is assigned to
V_m	mapping of message m to priority level-m
G_m	number of data bytes carried by message m
C_m	transmission time of message m
T_m	period or minimum inter-arrival time of message m
J_m	release jitter of messaage m
O_m	offset of message m
D_m	deadline of message m
R_m	worst-case response time of message m
St_m	absolute slack time of message m
S_m	relative slack of message m
S_{avg}	average of relative slacks of the system
W_m	queuing delay of message m
$lp(m)$	the set of messages with lower priority than m in message set Z
$hp(m)$	the set of messages with higher priority than m in message set Z
$lps(m,i)$	the set of messages that are sent by node i with lower priority than m
$hps(m,i)$	the set of messages that are sent by node i with lower priority than m
P^i	hyper-period of node i

2.2.2 Classical CAN Schedulability Analysis

In automotive applications, it is required that the ECUs such as the engine controller, the battery control unit or the brake control unit exchange data via CAN messages. That is, an application dependent set Z of CAN messages has to be transmitted on the CAN network.

In order to guarantee correct operation of the automotive application, it is required that each message meets its deadline, which has to be achieved by an appropriate assignment of the

unique message priorities. It has to be noted that, regarding CAN scheduling computations, it is merely important to consider the *order* of the message priorities rather than the actual priority values. Hence, in our work, we focus on the priority order represented by a map $V : m \rightarrow \{1, \ldots, M\}$ that assigns a unique *priority level* between 1 and M to each message in Z. We denote a priority assignment that ensures that each message meets its deadline as a *feasible CAN schedule* and the set of all feasible priority orders for a message set Z as $\mathcal{V}_S$.

We now summarize the classical method for the verification and computation of feasible CAN schedules that is the basis for our novel scheduling algorithms.

A given CAN schedulee is feasible if the *worst-case response time* R_m of each message m is smaller than the message deadline D_m. The classical CAN schedulability analysis as introduced in [7] and improved in [4] describes the worst-case response time of a message as made up of three elements [4]:

$$R_m = J_m + W_m + C_m \tag{2.3}$$

Here, J_m is the release jitter, W_m is the interference due to other messages, and C_m is the transmission time of message m. Since the release jitter is a given constant for each message, we assume that it is included in the deadline D_m of message m such that the equation simplifies to $R_m = W_m + C_m$. Interference due to higher priority messages which may win arbitration instead of message m is an important element contributing the queuing delay W_m [4]. Because of the impossibility of preempting message transmissions, a message is also subject to an initial blocking delay B_m from lower priority messages which is another element of queuing delay [14]. Maximum blocking time occurs when a lower priority message begins transmission just before message m is released.

$$B_m = \max_{k \in lp(m)} (C_k) \tag{2.4}$$

The concept of *busy period* is introduced by Lehoczky(1990) [15]. Busy period is a contiguous interval of time during which any lower priority message is not able to start transmission, starts at some time when a message of priority m or higher is queued ready for transmission and ends when the bus is idle. It is fundamental in worst-case response times analysis. The

busy period for priority $level - m$ starts with initial value $t_m^0 = C_m$ and finishes when $t_m^{n+1} = t_m^n$ and its equation is following:

$$t_m^{n+1} = B_m + \sum_{k \in hep(m)} \left\lceil \frac{t_m^n}{T_k} \right\rceil C_k \tag{2.5}$$

where $hep(m)$ is the set of messages with priority higher than equal to m, and $\lceil a/b \rceil$ is notation for the ceiling function which returns the smallest integer greater than or equal to a/b. $\left\lceil \frac{t_m^n}{T_k} \right\rceil$ indicates the number that message k which is in $hep(m)$ set is queued during t_m^n time. As the right hand side is a monotonic non-decreasing function of t_m, then the recurrence relation converges if the bus utilization U_m, for messages of priority m and higher, is less than 1 [4]:

$$U_m = \sum_{k \in hep(m)} \frac{C_k}{T_k} \tag{2.6}$$

There may be more than one message instances of m released during the busy period. In order to determine the worst-case response time, each instance is considered seperately. The maximum of these values give the worst-case response time [4]. The number of instances Q_m is given by:

$$Q_m = \left\lceil \frac{t_m}{T_m} \right\rceil \tag{2.7}$$

The time the qth instance starting transmission is given by:

$$W_m^{n+1} = B_m + qC_m + \sum_{k \in hp(m)} \left\lceil \frac{W_m^n}{T_k} \right\rceil C_k \tag{2.8}$$

The recurrence relation starts with $W_m^0(q) = B_m + qC_m$ and ends when $W_m^{n+1}(q) = W_m^n(q)$, or $J_m + W_m^{n+1}(q) - qT_m + C_m > D_m$ which implies message m is unschedulable. Worst-case response time for qth instance of message m is $R_m(q)$:

$$R_m(q) = W_m(q) - qT_m + C_m \tag{2.9}$$

The worst-case response time of message m:

$$R_m = \max_{q=0,\ldots,Q_m-1} (R_m(q)) \tag{2.10}$$

WCRT anlysis described in this section is for fixed-priority non-preemptive schedulings (FPNS). Worst-case response time can be computed if CAN priorities, message lengths, message periods are known. Hence, this analysis can be used to check existing schedule.

2.3 Generalized CAN Scheduling Algorithm

Based on the described schedulability analysis, the literature suggests an algorithm that successively computes a feasible CAN schedule if such schedule exists for a given message set Z [5, 4]. K. W. Schmidt [6] proposes the following generalized version of this algorithm .

Algorithm 1 Input: *message set Z, empty message* m_0*, set* $\mathcal{F} = \emptyset$*, initial priority level* $l := M$	
for each *priority level from* $l = M$ *to* $l = 1$	*1*
for each *message* $m \in S$	*2*
Compute W_m *assuming that all* $m' \in S - \{m\}$ *have higher priority than m*	*3*
if $W_m \leq D_m$	*4*
insert m in $\mathcal{F}$	*5*
if $\mathcal{F} = \emptyset$	*6*
return *Message set Z is not schedulable*	*7*
else	*8*
$m =$ `chooseMsgForLevel_l()`	*9*
if $m = m_0$	*10*
return *Message set Z is not schedulable*	*11*
else	*12*
Set $V(m) = l$*,* $\mathcal{F} = \emptyset$ *and* $S = S - \{m\}$	*13*
return *Feasible priority assignment o*	*14*

The algorithm uses the fact that *M priority levels*, i.e., one priority per message, have to be assigned for each message set Z. At each priority level, starting from the lowest priority level M, it is determined in line 3 and 4 which messages meet their deadline if all remaining unscheduled messages are assigned a higher priority level (this step is performed by applying the schedulability analysis in Section 2.2.2). One of these messages is then assigned the current priority level (line 8 – 10) and the algorithm proceeds to the next higher priority

level until all messages are assigned. Different from the previous literature [5, 4], generalized algorithm collects all schedulable messages at level l in the set $\mathcal{F}$. Then, the function `chooseMsgForLevel_l` is used to determine one message from $\mathcal{F}$ for level l (if no such message exists, a dummy message m_0 is returned and the message set is not schedulable). Consequently, different implementations of `chooseMsgForLevel_l` will yield different schedules. For example, the schedule according to [7, 4] is recovered if `chooseMsgForLevel_l` always returns the message that was first added to $\mathcal{F}$. The algorithm is guaranteed to find a feasible schedule if such schedule exists [5, 4] in at most $\frac{M(M+1)}{2}$ iterations since at each level l, at most l messages have to be checked. An important property of this algorithms is, that it finds a feasible schedule whenever such schedule exists [6], and hence yields the best scheduling result if messages IDs can be freely assigned.

CHAPTER 3

CAN Scheduling with Offsets

3.1 Offset Scheduling Description

CAN is an asynchronous bus which implies that any node may attempt to transmit a message without considering other nodes. Morover, any message may be queued at any time if there is not a criteria about release times of the messages. Hence, it has to be assumed that all higher priority messages are released at the same time in the worst case scenario. However, this assumption can be relaxed by controling the release times of messages of each node without introducing synchronization among the nodes. This can be achieved by introducing equally-spaced time windows in each node as is shown in Figure 3.1 for a window size of 2 ms. Then, the first instance of a message is released in a pre-specified window, that is called *offset*. For example, M_1 in Figure 3.1 is released by node N_1 with an offset of 0 and M_7 of node N_3 is released with offset 6 ms. Reference point of the offsets is the first time at which the node is ready to transmit. Following instances of the message are then sent periodically with accepting the first transmission as the time origin [13]. The assignment of offsets to all messages of a CAN network is denoted as an *offset schedule*.

After offsets are assigned to the messages, only messages that are assigned to the same window can be released at the same time. There is always the possibility that frames of any two or more messages coming from distinct nodes are released at the same time, inducing delays for some frames [13]. Depending on the clock skew of the different nodes, messages from all possible window combinations can be released at the same time. For example, it is possible that M_1 of N_1 and M_7 of N_3 are released at the same time due to clock skew of N_1 and N_3. However, in that case, for example message M_3 of N_3 cannot be released at the same time,

since it has a different offset. That is, offset scheduling reduces the number of messages that can be released at the same time.

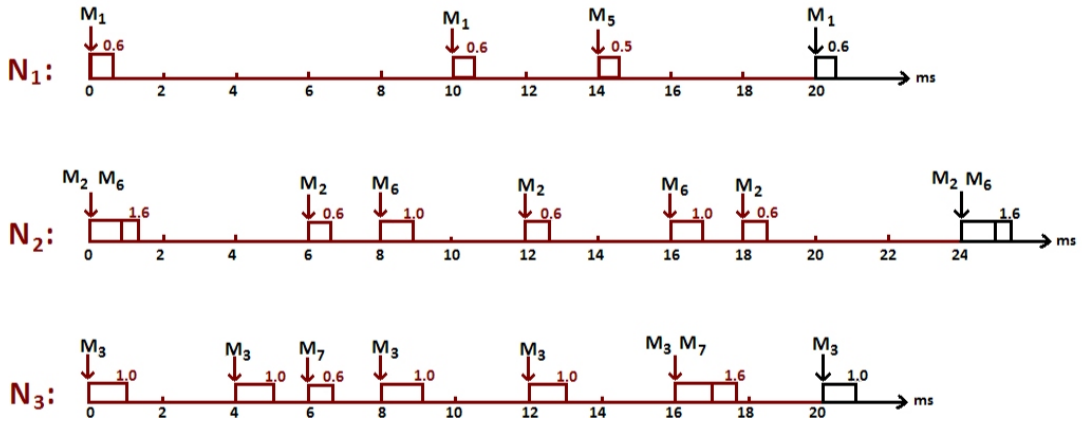

Figure 3.1: An example CAN Network with Offset Assigned Messages

A low-complexity algorithm for deciding offsets, which has good performances for typical automotive networks is proposed by N. Navet et. al. in [3]. Main purpose of this algorithm is to distribute the workload as uniformly as possible over time, in order to avoid synchronous releases leading to traffic peaks and thus to large frame response times. In other words, offset scheduling makes the transmissions as far apart as possible. This algorithm is explained in Seciton 3.2.

3.2 Standard Offset Scheduling Algorithm (SOSA)

Scheduling messages with offset is very beneficial in terms of worst-case response times. In this scheduling, the first instance of the message is released with delay, which is called *offset*. Subsequent instances are then sent periodically, with the first transmission as time origin. The challenge is to assign offsets for each message, which influences on the WCRT.

In the following chapter, we show that the complexity of finding optimum offsets is too much. A first approach in the literature is to spread the workload over time as much as possible, so that traffic peaks and large frame response times are avoided [3]. Offset assignment algorithm is applied to each node independently. We call this first aproach as *Standard Offset Scheduling Algorithm (SOSA)*. *SOSA* is a first heuristic approach proposed for offset scheduling in the

literature.

3.2.1 Design Hypothesis and Notations

The *SOSA* is introduced with the following design hypotheses that are adopted from the properties of practical applications.

- There are only a few distinct values for the periods (e.g. 5 to 10).
- The time is discrete with a certain granularity: the offsets of the messages, and their periods are multiples of g.

3.2.2 Description of the Algorithm

Aim of the algorithm is to find offset O_m of each message m in a node. In order to perform the algorithm, it is essential to know periods T_m of each message m in the node. The choise of the offset for message stream is made in the interval $[0, T_m)$. To spread the traffic over time, the offset of each message is chosen such that the release of its first message is "as far as possible" from other messages already scheduled. This is achieved by initially identifying the longest interval with the smallest workload, and then assigning offset for m in the middle of this interval.

Offsets are assigned based on an analysis performed over time interval $[0, T_{max})$, where T_{max} is the maximum of message periods in the node. The release times of the frames in the interval $[0, T_{max})$ are stored in an array A having T_{max}/g elements where i_{th} element of $A\,[i]$ is the set of frames released at possible release time i.

For each message m assigned to the node, initialy the least load, l_m, is calculated in the interval $[0, T_m)$. Then maximum of the least loaded intervals is found in the interval $[0, T_m)$, where least loaded intervals only comprise release times having a load equal to l_m. The first and last possible release time of the maximum of the least loaded intervals are denoted by I_k and E_k. Then, offset O_m is as the middle of the maximum of the least loaded intervals, the corresponding possible release time is denoted by a_k and $a_k = (I_k + E_k)/2$. Algorithm is applied to each node independently without considering messages of other nodes [3].

3.2.3 The Algorithm

Algorithm 2 Input: *message set Z, periods T_m of each message m, granularity length g*

Store the messages with increasing value of their period. 1

Calculate T_{max} 2

Create an empty release array having T_{max}/g elements 3

for each *message m_1 to m_n, Set offset for m_k* 4

Look for least load l_k in the interval $[0, T_k)$ 5

Look for one of the longest least loaded intervals in $[0, T_k)$ 6

Set the offset O_k in the middle of the 2^{nd} interval. 7

Update the release array A to store the frames of m_k in the interval $[0, T_{max})$ *:* 8

for each $i \in N$ *and* $a_k + i * (T_k/g) \leq (T_{max}/g)$

$A[r_k + i * T_k/g] = A[r_k + i * T_k/g] \cup m_{k,i} + 1$ 9

end

3.2.4 Application of the Algorithm

Consider a CAN Network having four different nodes and bandwidth of this network is 50 Kb/s. Messages of the network are described in the Table 3.1. In this example, assigning offsets only to the messages of N_1 is explained.

3 different messages are assigned to node N_1; $F = \{m_1, m_5, m_9\}$ with periods $T_1 = 10$, $T_5 = 20$ and $T_9 = 20$. Granularity of the system is $g = 2$ miliseconds. Initially, messages are stored by increasing value of their period: m_1, m_5 and m_9. T_{max} is the maximum of the periods which is 20; so array A has $\frac{T_{max}}{g} = 10$ elements.

First, the algorithm decides for offset of message m_1. Least load l_1 is 0 in the interval $[1, 5]$ implies I_1=1 and E_1=5 $a_1 = (1 + 5)/2 = 3$,so offset O_1 is $2 \cdot g$. Array A is updated as $A[3] = m_{1,1}$ and $A[8] = m_{1,2}$.

Then, the algorithm assigns offset to next message that is m_5. Least load l_5 is 0 and longest

Table 3.1: Example CAN Network Messages and Their Properties

Message	Node	Transmission Time, C_m in ms	Period, T_m in ms	Deadline, D_m in ms
m_1	N_1	0.6	10	5
m_5	N_1	0.5	20	10
m_9	N_1	1	20	10
m_2	N_2	0.6	6	20
m_6	N_2	1	8	20
m_{10}	N_2	0.5	12	40
m_3	N_3	1	4	50
m_7	N_3	0.8	10	7
m_{11}	N_3	0.6	20	5
m_4	N_4	1	4	20
m_8	N_4	0.5	24	20
m_{12}	N_4	1	24	10

least loaded interval is $[4, 7]$ implies I_5=4 and E_5=7 $a_5 = (4 + 7)/2 = 5$, so offset O_5 is $4 \cdot g$. So, array A is updated as $A[5] = m_{5,1}$.

For message m_9, least load l_9 is 0 and longest least loaded interval is $[9, 12]$ implies I_9=9 and E_9=12 $a_9 = (9 + 12)/2 = 10$, so offset O_9 is $9 \cdot g$. Hence, array A is updated as $A[10] = m_{9,1}$. Final release array of N_1 can be seen in Table 3.2.

Table 3.2: The release array A of node N_1

Possible Release Time (ms)	0	2	4	6	8	10	12	14	16	18
Possible Release Node, i	1	2	3	4	5	6	7	8	9	10
Release Array $A[i]$			$m_{1,1}$		$m_{5,1}$			$m_{1,2}$		$m_{9,1}$

Offsets of the messages of nodes N_2, N_3 and N_4 are assigned in a similar way as shown above. All offsets assigned to all messages in this network by applying *SOSA* can be seen in the Table 3.3.

SOSA proposed by Navet et. al. [3] is executed on each station independently without considering the streams of the other nodes. They show that *SOSA* provides better performance in terms of response times. However, it does not guarantee the schedulability of the system. The performance of offset assignments is evaluated over 1000 random sets of messages in [3]. They generate random message sets by using NETCARBENCH which is described in Section 5.4. The randomly generated networks in [3] have an average load equal to 35%,

Table 3.3: Example CAN Network Messages and Their Properties

Message	Node	Transmission Time, C_m in ms	Period, T_m in ms	Deadline, D_m in ms	Offset, O_m in ms
m_1	N_1	0.6	10	5	4
m_5	N_1	0.5	20	10	8
m_9	N_1	1	20	10	18
m_2	N_2	0.6	6	20	2
m_6	N_2	1	8	20	4
m_{10}	N_2	0.5	12	40	6
m_3	N_3	1	4	50	0
m_7	N_3	0.8	10	7	2
m_{11}	N_3	0.6	20	5	6
m_4	N_4	1	4	20	0
m_8	N_4	0.5	24	10	2
m_{12}	N_4	1	24	10	6

where *load* is percentage of bandwidth utilization by the messages in the whole network.

In their study, the WCRT of the messages are computed with the software NETCAR-Analyzer, first developed at INRIA, then taken over by the company RealTime-at-Work, which implements exact and very fast WCRT on CAN with offsets [3]. NETCAR-Analyzer is not used in this thesis, instead the algorithm proposed by Lei Du and Guoqing Xu in [9] is implemented to compute the WCRT of the messages.

SOSA improves *Classical Scheduling* in which offsets are not considered in the sense that feasible schedules can be found for higher loads. Related results that compare the performances can be seen in Section 5.6. However, *Standard Offset Scheduling Algorithm* does not consider what happens in other nodes and does not include any information from a worst-case response time analysis as described in this section.

3.3 Schedulability Analysis for Offset Scheduling

3.3.1 Description of the Algorithm

In order to guarantee that a message, especially a lower priority message, would meet its deadline, worst case response time (WCRT) has to be calculated. Classical WCRT analysis assumes that all messages can be generated at the same time. In offset scheduling, this is

avoided by controlling the release times of messages in the nodes even if the nodes are not synchronized. Hence, the WCRT analysis for conventional CAN as described in Section 2.2.2 has to be modified. We now describe the algorithm for the WCRT computation as presented in [9].

The worst-case response time of a message m is composed of two elements: the *maximum transmission* time C_m and the *queuing delay* W_m. The queuing delay W_m is made up of two items: *blocking* and *interference*. Blocking, B_m, is caused by lower priority messages, whose transmission continues when message m is released due to non-preempting nature of CAN. Interference is caused by higher-priority messages than m, that are transmitted before m can be.

In offset scheduling, more than one message can be released at the same time from different nodes. This causes lower priority messages to be interferred by higher priority messages [9]. The maximum amount of blocking occurs when a lower priority message starts transmission immediately before message m is queued and ready to be transmitted on the bus [13]. This message is called *blocking message*. So the next job is to find the possible blocking message instance named blocking message and message instances that are released simultaneously with message m and named *synchronized messages*. These synchronized messages are obtained after an iterative process, which is explained in later.

Initially, the synchronized message is chosen for each node different from N_m such that it has the longest transmission time among the set of messages that are sent by that node with higher priority than m. The initial queuing delay, W_m^0, is calculated with these interferences; however these synchronized messages may change in later.

Message instance candidates for blocking and synchronized messages are looked in the interval called *feasible interval* [9]. This interval starts at the maximum release offset of $hps(m,i)$, O^i_{max}, and ends at $O^i_{max}+P_i$ where P_i is the *hyper-period* that is calculated as least common multiple of the periods of the messages in $hps(m,i)$. Property of feasible interval is that the instances of messages in the next hyper-period are the replicate of instances in the feasible interval.

Figure 3.2 illustrates feasible interval and hyper-period of message m_8 defined in the network which is described in Section 3.2.4. Higher priority messages released from node N_1,

$hps(m_8, 1)$, are message m_1 and message m_5. Maximum release offset of $hps(m_8, 1)$, O^1_{max}, is 8 ms and length of hyper-period P_1 is the least common multiple of the message periods, $lcm(10, 20)$, which is 20. As can be seen from Figure 3.2, lighter gray painted area indicates the feasible interval. Feasible interval starts at O^1_{max}=8 ms and ends at $O^1_{max} + P_1$ =28 ms. Instances of messages in darker gray painted area is the next hyper-period and the replicate of the feasible interval.

Possible Release Time (ms)	0	2	4	6	8	10	12	14	16	18	20	22	24	26
Possible Release Index, *i*	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Release Array A[i]			m_1		m_5			m_1				m_1		

28	30	32	34	36	38	40	42	44	46
15	16	17	18	19	20	21	22	23	24
m_5			m_1				m_1		

Figure 3.2: Illustration of hyper-period and feasible interval

Assuming that all higher priority messages which has the longest transmission time of the node are released simultaneously from other nodes. Since the aim is to calculate worst case response time, maximum interference of node i for message m and initial synchronized window are found at each node different than N_m among $hps(m, i)$.

$$C_{max}(i) = \max_{k \in hps(m,i)} (C_k) \tag{3.1}$$

Considering the network described in Section 3.2.4, worst-case response time of message m_8 is analyzed as an example. In the example, maximum interferences of the nodes for message m_8 are found as $C_{max}(1) = 0.6$ ms, $C_{max}(2) = 1$ ms and $C_{max}(3) = 1$ ms.

Blocking message of node i for message m, which is generated from different node than N_m and has the longest transmission time of the node is found among the lower priority messages.

$$B_{max}(i) = \max_{k \in lps(m,i)} (C_k) \tag{3.2}$$

In the example, blocking messages of each node for message m_8 are like following: $B_{max}(1)$

= 1 ms, $B_{max}(2)$ = 0.5 ms and $B_{max}(3)$ = 0.6 ms.

In the following term, maximum blocking time is calculated. While calculating this term, synchronized message, $C_{max}(i)$, is subtracted since it will be added while calculating queuing delay; so that synchronized message is not taken into account for the node from where blocking message is released.

$$B_m = \max_{1 \leq i \leq Nmax, i \neq Nm} (B_{max}(i) - C_{max}(i)) \tag{3.3}$$

If B_m is negative, this means that lower priority messages do not contribute to worst-case response time of message m, so:

$$B_m = 0$$

Blocking for message m_8, B_{m_8}, is found like below:

$$B_{m_8} = max((1 - 0.6), (0.5 - 1), (0.6 - 1)) = 0.4ms$$

In the example, blocking term is obtained from node N_1 for message m_8, so initial contribution of node N_1 to qeuing delay of message m_8 becomes $Ctrb_1^0 = B_{max}(1)$ = 1 ms.

Higher priority messages released simultaneously with message m from the same node N_m may also interfere with message m and worst-case of this interference is symbolized by C. These interference messages are denoted by $eqr(m, i)$

$$C = max_{i=1...\lfloor P/T_m \rfloor} \left(\sum_{k \in eqr(m,i)} C_k \right) \tag{3.4}$$

In the example, since there is no higher priority message than message m_8 that are released simultaneously with message m_8 from node N_4, which can be seen from the Figure 3.3; so $C = 0$.

Real contributions of the nodes are obtained after an iterative process, which is explained in later. The contribution of node i at n^{th} iteration is notated by $Ctrb_i^n$. Each node i may

contribute to the initial queuing delay of m by a blocking message or synchronized messages or higher priority messages released simultaneously with message m from the same node N_m.

$$Ctrb_i^0 = \begin{cases} C & \text{if } i = N_m; \\ B_{max}(i) & \text{if } i \neq N_m \text{ and node } i \text{ is where blocking message is assigned;} \\ C_{max}(i) & \text{else;} \end{cases}$$

For the situation of the example, initial contributions of N_1 is due to a blocking message and $Ctrb_1^0$ = 1 ms. Initial contributions of N_2 and N_3 are due to synchronized messages; $Ctrb_2^0$ = 1 ms and $Ctrb_3^0$ = 1 ms. N_4 is the node where message m_8 is assigned to; so it may only contribute with term C. However, it does not contribute at this example, in other words $Ctrb_4^0$ = 0.

The initial queuing delay, W_m^0, is the sum of the blocking time, interference of higher priority messages released from other nodes and interference of message instances released simultaneously with message m from the same node.

$$W_m^0 = C + B_m + \sum_{1 \leq i \leq N_{max}, i \neq N_m} C_{max}(i) \tag{3.5}$$

In the example, initial queuing delay of message m_8 is calculated as below:

$$W_{m_8}^0 = B_{m_8} + C_{max}(1) + C_{max}(2) + C_{max}(3) = 0.6 + 1 + 1 + 0.4 = 3ms$$

Initially, it as assumed that contribution of each node is obtained from the messages having highest transmission times. However, worst case response times may occur at different situations; so maximum contribution term is searched at each node with an iterative process. Initial step is finished when initial queuing delay, W_m^0, is calculated. By using initial queuing delay, W_m^0, real queuing delay is obtained after an iterative process. Stopping condition of the iteration process at k_{th} iteration is either when $W_m^k = W_m^{k-1}$ or when queuing delay becomes greater then the deadline, $D_m < W_m^k$.

$$W_m^k = W_m^0 + \sum_{1 \leq i \leq N_{max}} Ctrb_i^k \tag{3.6}$$

Now, by using initial queuing delay $w^0_{m_8}$, contribution terms of each node are calculated at each iteration and they are added to the queuing delay. Iteration stops either when queuing delay does not change or queuing delay reaches to deadline of the message.

In the example, let's examine the contributions of each node in detail. Initial contribution of node N_1, $Ctrb^0_1$, is obtained from a blocking message that is shown above. Now, for each instance in the feasible interval of N_1 how the queuing delay is affected is examined as if that instance is the synchronized message. Feasible intervals of each node for message m_8 can be seen in the Figure 3.3. In the figure, $J_{i,k}$ indicates the k^{th} possible synchronized instance of node i. As can be seen from the Figure 3.3, any instance in the feasible of node N_1 decreases the queuing delay if they were the synchronized messages. Since node N_1 does not make positive contribution at this iteration, then $Ctrb^1_1$ is 0.

Initial contribution of node N_2, $Ctrb^2_0$, is 1 ms. If $J_{2,1}$ was the synchronized message from node N_2, it wouldn't make an extra contribution to the queuing delay. If $J_{2,2}$ was chosen, queuing delay would decrease -0.4 ms. If $J_{2,3}$ was the synchronized message from node N_2, $J_{2,4}$ would also be included in the queuing delay; so $J_{2,3}$ would contribute 0.6 ms. $J_{2,4}$ would also decrease the queuing delay -0.4 ms if that was the synchronized message from N_2. $J_{2,5}$ and $J_{2,6}$ would contribute 0.6 ms to the queuing delay. As a result, contribution of node N_2 at this iteration, $Ctrb^2_1$, is the maximum contribution of the instances which is 0.6 ms.

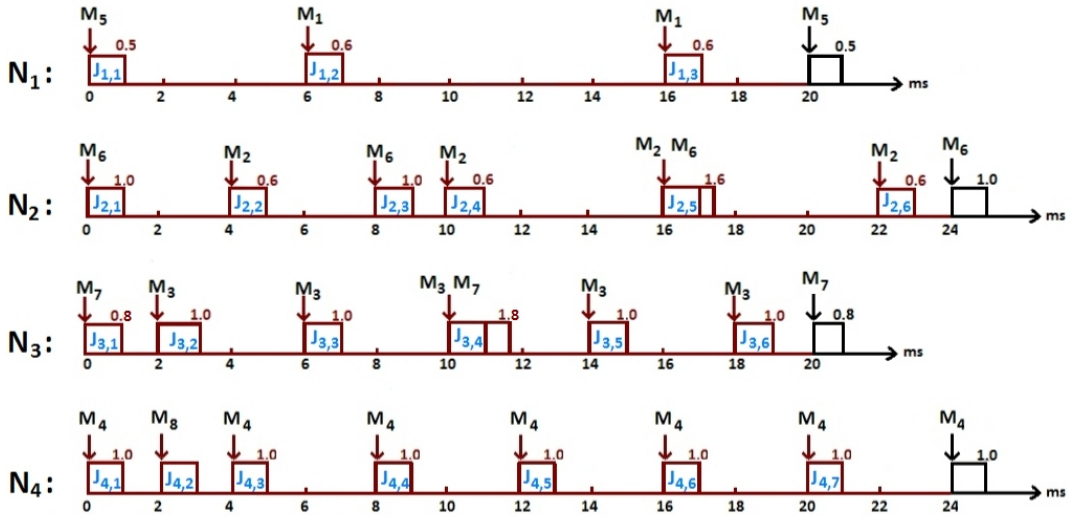

Figure 3.3: Possible Synchronized Messages of message m_8

Node N_3 initially contributes to the queuing delay of message m_8, $Ctrb_0^3$, 1 ms. $J_{3,1}$, $J_{3,4}$ and $J_{3,6}$ would make extra 0.8 ms contribution while $J_{3,2}$, $J_{3,3}$ and $J_{3,5}$ does not make any contribution to queuing delay. Hence, contribution of node N_3 at this iteration, $Ctrb_1^3$, is 0.8 ms.

For node N_4, $J_{4,2}$ is the only instance to be interested in because that is the only window where message m_8 is released. Instance $J_{4,2}$ would also be included in at this iteration due to length of initial queuing delay. Hence N_4 contributes 1 ms to the queuing delay at this iteration.

Contributions of each node for first iteration are found, so new queuing delay can be calculated. $w_{m_8}^1 = w_{m_8}^0 + Ctrb_1^1 + Ctrb_1^2 + Ctrb_1^3 + Ctrb_1^4$ = 3+0+0.6+0.8+1 = 5.4 ms. Since $w_{m_8}^1 > w_{m_8}^0$, algorithm continues with next iteration.

Table 3.4: Iteration 1

Node	Instance	Contribution of Instance	Contribution of Node
N_1	$J_{1,1}$	-0.5 ms	$Ctrb_1^1$ is 0
N_1	$J_{1,2}$	-0.4 ms	
N_1	$J_{1,3}$	-0.4 ms	
N_2	$J_{2,1}$	0	$Ctrb_1^2$ is 0.6 ms
N_2	$J_{2,2}$	-0.4 ms	
N_2	$J_{2,3}$	0.6 ms	
N_2	$J_{2,4}$	-0.4 ms	
N_2	$J_{2,5}$	0.6 ms	
N_2	$J_{2,6}$	0.6 ms	
N_3	$J_{3,1}$	0.8 ms	$Ctrb_1^3$ is 0.8 ms
N_3	$J_{3,2}$	0	
N_3	$J_{3,3}$	0	
N_3	$J_{3,4}$	0.8 ms	
N_3	$J_{3,5}$	0	
N_3	$J_{3,6}$	0.8 ms	
N_4	$J_{4,1}$	1 ms	$Ctrb_1^4$ is 1 ms

$w_{m_8}^1 = w_{m_8}^0 + Ctrb_1^1 + Ctrb_1^2 + Ctrb_1^3 + Ctrb_1^4$ = 3+0+0.6+0.8+1 = 5.4 ms

Table 3.5: Iteration 2

Node	Instance	Contribution of Instance	Contribution of Node
N_1	$J_{1,1}$	-0.5 ms	$Ctrb_2^1$ is 0.1 ms
N_1	$J_{1,2}$	-0.4 ms	
N_1	$J_{1,3}$	0.1 ms	
N_2	$J_{2,1}$	0.6 ms	$Ctrb_2^2$ is 0.6 ms
N_2	$J_{2,2}$	0.6 ms	
N_2	$J_{2,3}$	0.6 ms	
N_2	$J_{2,4}$	-0.4 ms	
N_2	$J_{2,5}$	0.6 ms	
N_2	$J_{2,6}$	0.6 ms	
N_3	$J_{3,1}$	0.8 ms	$Ctrb_2^3$ is 1.8 ms
N_3	$J_{3,2}$	1 ms	
N_3	$J_{3,3}$	1.8 ms	
N_3	$J_{3,4}$	1.8 ms	
N_3	$J_{3,5}$	1 ms	
N_3	$J_{3,6}$	1.8 ms	
N_4	$J_{4,1}$	1 ms	$Ctrb_2^4$ is 1 m

$w_{m_8}^2 = w_{m_8}^0 + Ctrb_2^1 + Ctrb_2^2 + Ctrb_2^3 + Ctrb_2^4$ = 3+0.1+0.6+1.8+1 = 6.5 ms

Table 3.6: Iteration 3

Node	Instance	Contribution of Instance	Contribution of Node
N_1	$J_{1,1}$	0.1 ms	$Ctrb_3^1$ is 0.1 ms
N_1	$J_{1,2}$	-0.4 ms	
N_1	$J_{1,3}$	0.1 ms	
N_2	$J_{2,1}$	0.6 ms	$Ctrb_3^2$ is 1.2 ms
N_2	$J_{2,2}$	1.2 ms	
N_2	$J_{2,3}$	0.6 ms	
N_2	$J_{2,4}$	-0.4 ms	
N_2	$J_{2,5}$	1.2 ms	
N_2	$J_{2,6}$	1.2 ms	
N_3	$J_{3,1}$	0.8 ms	$Ctrb_3^3$ is 1.8 ms
N_3	$J_{3,2}$	1 ms	
N_3	$J_{3,3}$	1.8 ms	
N_3	$J_{3,4}$	1.8 ms	
N_3	$J_{3,5}$	1 ms	
N_3	$J_{3,6}$	1.8 ms	
N_4	$J_{4,1}$	2 ms	$Ctrb_3^4$ is 2 ms

$w_{m_8}^3 = w_{m_8}^0 + Ctrb_3^1 + Ctrb_3^2 + Ctrb_3^3 + Ctrb_3^4$ = 3+0.1+1.2+1.8+2 = 8.1 ms

Table 3.7: Iteration 4

Node	Instance	Contribution of Instance	Contribution of Node
N_1	$J_{1,1}$	0.1 ms	$Ctrb_4^1$ is 0.1 ms
N_1	$J_{1,2}$	-0.4 ms	
N_1	$J_{1,3}$	0.1	
N_2	$J_{2,1}$	0.6 ms	$Ctrb_4^2$ is 1.2 ms
N_2	$J_{2,2}$	1.2 ms	
N_2	$J_{2,3}$	0.6 ms	
N_2	$J_{2,4}$	1.2 ms	
N_2	$J_{2,5}$	1.2 ms	
N_2	$J_{2,6}$	1.2 ms	
N_3	$J_{3,1}$	1.8 ms	$Ctrb_4^3$ is 2.8 ms
N_3	$J_{3,2}$	1 ms	
N_3	$J_{3,3}$	2.8 ms	
N_3	$J_{3,4}$	2.8 ms	
N_3	$J_{3,5}$	2.8 ms	
N_3	$J_{3,6}$	2.8 ms	
N_4	$J_{4,1}$	2 ms	$Ctrb_4^4$ is 2 ms

$w_{m_8}^4 = w_{m_8}^0 + Ctrb_4^1 + Ctrb_4^2 + Ctrb_4^3 + Ctrb_4^4$ = 3+0.1+1.2+2.8+2 = 9.1 ms

Table 3.8: Iteration 5

Node	Instance	Contribution of Instance	Contribution of Node
N_1	$J_{1,1}$	0.1 ms	$Ctrb_5^1$ is 0
N_1	$J_{1,2}$	-0.4 ms	
N_1	$J_{1,3}$	0.1 ms	
N_2	$J_{2,1}$	1.6 ms	$Ctrb_5^2$ is 2.2 ms
N_2	$J_{2,2}$	1.2 ms	
N_2	$J_{2,3}$	2.2 ms	
N_2	$J_{2,4}$	1.2 ms	
N_2	$J_{2,5}$	2.2 ms	
N_2	$J_{2,6}$	1.2 ms	
N_3	$J_{3,1}$	1.8 ms	$Ctrb_5^3$ is 2.8 ms
N_3	$J_{3,2}$	1 ms	
N_3	$J_{3,3}$	2.8 ms	
N_3	$J_{3,4}$	2.8 ms	
N_3	$J_{3,5}$	2.8 ms	
N_3	$J_{3,6}$	2.8 ms	
N_4	$J_{4,1}$	2 ms	$Ctrb_5^4$ is 2 ms

$w_{m_8}^5 = w_{m_8}^0 + Ctrb_5^1 + Ctrb_5^2 + Ctrb_5^3 + Ctrb_5^4$ = 3+0.1+2.2+2.8+2 = 10.1 ms

Since $w_{m_8}^5 = 10.1$ ms $> D_{m_8} = 10$ ms , iteration stops here; message m_8 is not schedulable. So, it can be concluded that system is also not schedulable.

3.3.2 The Algorithm

The worst-case response time analysis for offset scheduling algorithm described above is summarized in the following procedure.

Algorithm 3 Input: *Priority, Period T_m, Transmission time C_m and Offset O_m of each message m;*

for each *node i calculate*

Calculate, T^i_{max}, period of the message which has longest period

Calculate, P_i, hyper-period

end

end *message m, m = 1 to M_m*

for each *node i = 1 to N_{max}*

Find $C_{max}(i)$ and $B_{max}(i)$

Find release time of instances in feasible interval

end

Calculate B_m, w^0_m and $Ctrb^0_i$

$w^n_m = 0$

while $w^n_m \neq w^0_m$

$w^n_m = w^0_m$

for each *node i = 1 to N_{max}*

for each *instance j, j = 1 to $Nins(m, i)$ in feasible interval in node i*

Compute $Ctrb_j$ for instance j with initial value w^n_m

end

$Ctrb^n_i = max_{1 \leq j \leq N_{ins}(m,i)}(Ctrb_j)$

return the synchronized message

$w^n_m = w^0_m + Ctrb^n_i$

end

end

$R_m = w_m + C_m$

return R_m

end

The worst-case response time analysis for offset scheduling algorithm described in this section is implemented and used to analyze the performances of the offset scheduling algorithms in this thesis.

CHAPTER 4

Developed Algorithms for Offset Scheduling

4.1 Problems and Challenges

4.1.1 Problem

In this thesis, we aim to find a feasible offset schedule for a given message set on CAN whenever it exists. *Standard Offset Scheduling Algorithm (SOSA)* assigns offsets as described in Section 3.2 by trying to distribute the workload as uniformly as possible over time for each node without including any worst-case response time analysis. As the main contribution of this thesis, we develop two different algorithms that improve the performances of *SOSA* in the sense of schedulability at higher loads. The first algorithm is denoted as *Local Neighborhood Search Algorithm (LNSA)*. It starts from the results of *SOSA* and tries to vary the message offsets, while retaining best solution. The second algorithm is a genetic algorithm that evolves along random variations of message offsets.

4.1.2 Challenges

In offset scheduling, the search space is too large to find a feasible schedule for message sets of realistic sizes. In order to illustrate this claim, consider a message set in which there are M messages released from N different nodes. T_m/g is the number of possible offsets that can be assigned to message m, where T_m is the period of message m and g is the length of a window in the offset schedule and calculated as the gcd of the periods in the system. Each offset schedule is defined by the assignment of one unique offset to each message. Hence, the number of different possible schedules for a message set is given by the product of the

number of offsets for each message. We denote this number as $SCHD_{max}$, and compute:

$$SCHD_{max} = \prod_{i=1}^{N_{max}} T_i/g \tag{4.1}$$

If O_{max} denotes the maximum number of possible offsets, then the size of the search space is bounded by $O_max^N ax$. Accordingly, an exhaustive search for a feasible schedule has a complexity $O((O_{max})^{N_{max}})$ which is exponential in the number of messages.

The main objective of offset scheduling is to assign message offsets in order to achieve a feasible schedule. Since the search space is very large, any practically applicable scheduling algorithm must employ a heuristic seach strategy in order to avoid scanning the whole search space. For instance, for a small message set composed of 10 messages whose periods are 10 ms and granularity of the system is 2 ms; there are $(10/2)^{10}$, 9765625, different possible schedulings. In practice, number of messages is actually order of 50 or more; so it is inpractical to analyze all possible schedulings.

The heuristic of the *SOSA* is to assign offsets such that as few messages of each node are released at the same time. This heuristic is computationally very efficient, since it only requires computations on the small message sets for each invdividual node. However, the method does not consider the interference and blocking caused by messages of other nodes as discussed in Section 3.3. Hence, as is confirmed in Section 5.6, a satisfactory performance of this algorithm cannot be expected.

The aim of this thesis is the development of offset scheduling algorithms for CAN, that incorporate information about the dependencies between nodes. As will be shown in Chapter 5, neglecting these dependencies, as is done in the *SOSA*, significantly reduces the network load, where feasible schedules can be found. The main idea of the thesis is to introduce the slack for each message, and to reformulate the offset scheduling problem as a constraint optimization problem that maximizes the sum of message slacks, while ensuring that all slacks remain positive. The following sections elaborate this idea.

4.1.3 Offset Scheduling as Optimization Problem

If a message meets its deadline despite a worst-case release pattern of other messages, that message is schedulable. A system is schedulable if all messages in the system are schedulable. The main purpose of the scheduling algorithms are to guarantee that all messages arrive at the destination node before their deadlines. So, the most important constraint for the scheduling algorithms is the schedulability.

$$\forall m = 1, \ldots, M : R_m \leq D_m$$

Absolute slack time of message m is denoted by St_m and is the difference between the deadline and the worst-case response time; $St_m = D_m - R_m$. If absolute slack time of a message is low, this indicates that message is close to be unschedulable. The higher the absolute slack time, the less critical that message to be unschedulable. So, it is desirable to obtain high absolute slack times for the messages. In order to use absolute slack time of message with long deadline and absolute slack time of message with small deadline at the same time, it is more preferable to use relative slack slack of the messages which is the ratio of slack time of a message to its deadline, $S_m = St_m/D_m$. In order to compare the performances of solutions; average of relative slacks of the system, S_{avg} is used.

$$S_{avg} = \left(\sum_{1 \leq m \leq M} S_i \right) / M \tag{4.2}$$

If S^i_{avg} denotes the average of relative slacks of the system for the schedule $SCHD_i$, then the constraint optimization problem can be formulated like following:

$$\max_{1,\ldots,SCHD_{max}} \left(S_{avg} \right) \qquad \textit{subject to} \qquad \forall m = 1, \ldots, M : S_m \geq 0$$

where S_{avg} is the objective function as defined in (4.2), $S_m \geq 0$ are the constraints and the maximization is performed over all possible schedules $1, \ldots, SCHD_{max}$.

In the next sections, we present two heuristic optimization strategies in order to estimate an optimal solution. It has to be noted that our main goal is not finding a global optimum for

S_avg but finding a large value for S_avg while meeting the schedulability constraint.

4.2 Local Neighborhood Search Algorithm (LNSA)

4.2.1 Description of LNSA

In the thesis, two different heuristics are proposed to solve the optimization problem. The first one is named as *Local Neighborhood Search Algorithm (LNSA)* which starts from the solution of *SOSA* with using different message sorting strategies.

SOSA does not care about dependencies among nodes but schedules each node independently. From point of view of node i where message m is released, contributions of other nodes to WCRT of message m are unpredictable. So, distributing the workload as uniformly as possible over time may not give good solution every time. In our approach, we use the *SOSA* as a basis and then do the WCRT analysis to successively verify schedulability and then change the schedule by keeping the best solution.

Initially, *LNSA* assigns offsets to the messages with *SOSA*. Then, it sorts the messages in the message set according to three optional strategies. These strategies are by slacks, by IDs and by deadlines of the messages. When messages are sorted by slacks of the messages, algorithm starts from the message having minimum slack according to the *SOSA*. Small slack indicates that message is critical to be unschedulable and zero slack states that message is unschedulable.

While messages are sorted according to IDs of the messages, *LNSA* starts analyzing the messages beginning from maximum ID message; i.e. lowest priority message. Sorting the messages by deadlines indicates that messages are put in the order as message having smallest deadline is used first. In order to differentiate which sorting strategy is used with the algorithm, name of the sorting strategy is written in the paranthesis after *LNSA*. For instance, *LNSA (by ID)* indicates that messages are sorted according to IDs of the messages in *LNSA*.

Algorithm visits each message once in the order of how they are sorted. When a message is visited, algortihm does the WCRT analysis for all possible offsets of that message while keeping offsets of other messages the same. Then, slacks of all messages in the message set are summed and offset which gives the maximum of the sum of slacks is assigned to

the current message. This means that schedule is modified to a new schedule and algorithm continues with the modified schedule for the remainig messages.

The procedure described above is summarized in the following algorithm. If O_{max} denotes the maximum number of possible offsets, then complexity of algorithm is $O(O_{max} \cdot M)$.

Algorithm 4 Input: *message set Z*

Do offset scheduling with SOSA *for Z* 1

Sort the messages by ID or Slack or Deadline 2

stdSumSlack = sum of slacks of Standard Offset Schedule 3

maxSumSlack = stdSumSlack 4

for each *message m in sorted message set Z* 5

  currentSumSlack = 0 6

  for each *possible offset o of message m* 7

    assign the offset to current message 8

    do WCRT analysis and remember the slack S_m of each message m 9

    for each *message m in set Z* 10

      currentSumSlack = currentSumSlack + S_m 11

    end 12

    if *currentSumSlack > maxSumSlack* 13

      maxSumSlack = currentSumSlack 14

      offset of the message m, $O_m = o$ 15

    end 16

  end 17

end 18

4.2.2 Illustration of LNSA

Consider a network with 3 nodes N_1, N_2 and N_3. Granularity, g, of the system is 200 us. 3 different messages are assigned to N_1; which are m_1,m_2,m_3 with periods $T_1 = 600ms$, $T_2 = 1200ms$, $T_3 = 1200ms$. *SOSA* assigns offsets to these message as described in Section 3.2; $O_1 = 200ms$, $O_2 = 400ms$, $O_3 = 0ms$.

LNSA initially analyzes WCRTs of each message in the system and calculates the sum of slacks of all messages then saves the result in *stdSumSlack* variable. Suppose that sum of slacks of the system that *SOSA* assigned its offsets is initially calculated as *stdSumSlack* = 8050. So, *maxSumSlack* is initialized to 8050. Assume that all messages in the system are

sorted with respect to their periods.

Messages are analyzed in the order of how they are sorted. Without considering other two nodes and their messages in the system, assume that present order is for messaage m_2. Message m_2 has T_2/g=1200/200=6 different possible offsets which are $O_2^1 = 0$, $O_2^2 = 200$, $O_2^3 = 400$, $O_2^4 = 600$, $O_2^5 = 800$ and $O_2^6 = 1000$. Each possible offset is assigned to message m_2 one-by-one, and WCRTs of m_2 and rest of the messages in the system are analyzed. Then, slacks of all messages are summed for each offset. If the sum of slacks is greater than *stdSumSlack*, then that offset is included in possible new offset set. Assume that sum of slacks are calculated like 8000, 8100, 8050, 7950, 8250 and 8050 for each possible offset of m_2. It is seen that when offsets O_2^2 or O_2^5 are assigned to message m_2, better performance is obtained. New offset is chosen as the offset resulting in the maximum of the sum of slacks value if greater than *stdSumSlack*. So; new offset of message m_2 is O_2^5 implying $O_2 = 800$ ms and *maxSumSlack* becomes 8250.

Message m_3 also has 6 different possible offsets. Assume that *maxSumSlack* is still 8250 and following sum of slack values are calculated for corresponding offsets: 8250, 7950, 8000, 8050, 7800 and 7900. It can be seen that none of these values is greater than *maxSumSlack*. Hence, offsets of message m_3 remains same, which is $O_3 = 0$ ms.

Suppose that *maxSumSlack* is 8400 when present order is for messaage m_1 and m_1 is the final message. m_1 has $600/200 = 3$ different possible offsets and sum of slack values are calculated for corresponding offsets: 8450, 8450 and 8000. O_1^1 and O_1^2 are both greater than *maxSumSlack*, but O_1^1 is chosen as the new offset to be assigned to m_1 because it comes earlier. So, $O_1 = 0$.

As illustrated in the example above, main goal is to find a large value for S_{avg}, which implies the decrease in the total WCRT of the system. As shown in Section 5.6, this also results in performance boost in terms of schedulability. On the contrary, LNSA changes one offset at a time to find a better schedule; however, a better solution may be obtained by modifying two or more offsets at the same time.

4.3 Genetic Algorithm Approach

4.3.1 General Description

Genetic Algorithm (GA) is an optimization algorithm that mimics the process of natural evolution, such as inheritance, mutation, selection and crossover. *GA* simulates processes in natural system necessary for evolution, specifically survival of the fittest. *GA*s show good performance even for large search spaces, and potentially find good solutions rapidly.

GA starts with a randomly generated *set of solutions* called *population*. In order to generate a new population, solutions from the previous population are taken and processed. In particular, the fitness of every individual in the population is evaluated. Individuals having better fitness are selected as parents to form new individuals. Genetic operators such as mutation and crossover are applied to evolve the solutions in order to find the best one. The new population is used in the next iteration of the algorithm. The *GA* terminates either if a satisfactory solution is obtained or if the maximum number of iterations is reached. It has to be noted that, if the algorithm has terminated due to maximum number of iterations, a satisfactory solution may or may not have been obtained. The general procedure for a *GA* is shown in Algorithm 5.

Algorithm 5 Input:Offset Schedule, mutation probability, crossover probability	
Generate random population of n individuals	*1*
Evaluate the fitness of each individual in the population	*2*
Create new population by repeating following steps until the new population is complete	*3*
Select two parent individuals from the population according to their fitness.	*4*
Cross over the parents with a crossover probability to generate new children.	*5*
Make mutations with a mutation probability.	*6*
Put new children into the population.	*7*
Use new generated population for a further run of algorithm	*8*
if *the end condition is satisfied*	*9*
stop and return the best solution in current population	*10*
else	
Go to step 2	*11*

In order to use *GA*, some parameters have to be adjusted before. *maxIterationNo* is the maximum number of iterations that *GA* can make. Population size is denoted by *populationNo*.

Maximum number of child schedules that can be generated from parents at each iteration is indicated by *maxChildrenNo*. *parentNo* indicates the number of parent schedules that are chosen among the schedules and have the best fitness. Amount of the offsets to be mutated is determined by some ratio of messages in the schedule, which is called *mutation probability (ratio)* and denoted by *mutationRatio*. These messages are chosen randomly among all messages. *Crossover probability (ratio)* describes the ratio of offsets obtained from one of the parents and remaining offsets are obtained from the other parent. *crossover probability (ratio)* is denoted as *crossoverRatio*.

4.3.2 Genetic Algorithm for Offset Scheduling (GAOS)

Since the search space is too large for offset scheduling, *GAOS* is applied as an alternative to *LNSA*s defined in Section 4.2. In the *LNSA*s, we always change a single offset and we always keep the "best" solution. Now for *GAOS*, we keep a set of "fittest" solutions and evolve the optimization from there. It has to be noted that we do not use the *GAOS* in order to find an optimal feasible solution, but we use the *GAOS* in order to determine search directions that move solution schedules from the infeasible to feasible solutions. Hence, the *GAOS* is applied to search in the infeasible space by maximizing the fitness of individuals to move into the feasible space. If a feasible solution is found, *GAOS* stops.

In offset scheduling case, we define an individual as an offset schedule which is composed of assigned offsets for each message in the message set; i.e. individuals of the population are the offset schedules. In our solution, we generally use the solution of *SOSA* in order to generate initial population. Precisely, the *Standard Offset Schedule* constitutes one individual of the initial population, and other individuals of the initial population are generated by mutations of the *Standard Offset Schedule*. Here, a mutation is obtained by randomly assigning offsets to chosen message among all possible offsets of that message.

Figure 4.1 (a) shows an example schedule generated by *SOSA*. Schedule is composed of fifteen messages and offsets assigned to these messages. (b), (c) and (d) are the other schedules obtained by applying mutations to *Standard Offset Schedule*. In this example, *mutation ratio* is 20% which means 3 of 15 messages are chosen randomly to apply mutation.

Fitness of each schedule in the population is evaluated by doing worst-case response time

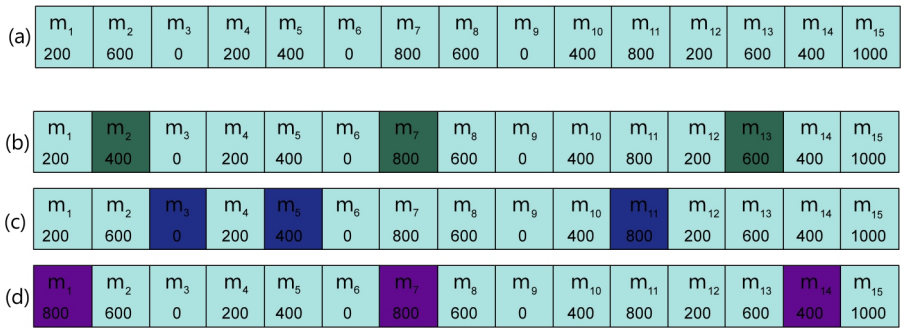

Figure 4.1: Example initial population to use in Genetic Algorithm

analysis which gives performance metrics of a schedule which are the schedulability and sum of slack times of a schedule. Then, *parentNo* number of parent schedules having the best fitness are chosen from the population in order to generate new individuals. In the implementation of *GAOS*, *parentNo* is adjusted to 2. Number of new indidual schedules are determined randomly between one and *maxChildrenNo* at each iteration. *maxChildrenNo* is set to 5 in the implementation of *GAOS*. Offsets of new children schedules are initially assigned by crossing over the parents with a *crossover probability*. Then, some number of message offsets which is described by *mutation ratio* are mutated. Fitness of new child schedules are evaluated, and if a schedulable solution is obtained the algorithm stops; otherwise weakest solutions are eliminated to keep the population size at *populationNo* and algorithms continues at most for predefined *maximum iteration number*.

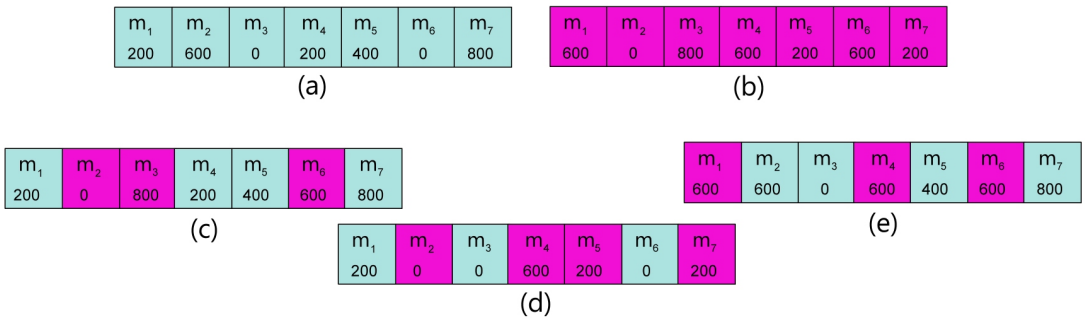

Figure 4.2: Illustration of Crossover Used for Generating New Schedules. (a) and (b) are the parent schedules, and (c), (d) and (e) are the child schedules generated from (a) and (b)

How parent schedules are crossovered to generate new child schedules are illustrated in Figure 4.2. (a) and (b) are the parent schedules which are chosen from the population because they have the best fitness. In this example, three child schedules are generated which are (c), (d) and (e) and *crossover ratio* is 3/7. As shown, each message offset of child sched-

ules is produced by randomly choosing the parent schedule and then assigning its offset to corresponding message.

Figure 4.3 shows an example of mutation that is applied to the schedules. Initially, number of messages that are used for mutation is determined according to the mutation ratio. In this example, one of the seven messages is chosen for mutation. This message is chosen randomly from the message set. The mutation is performed by randomly assigning a new offset to the message from the possible offsets.

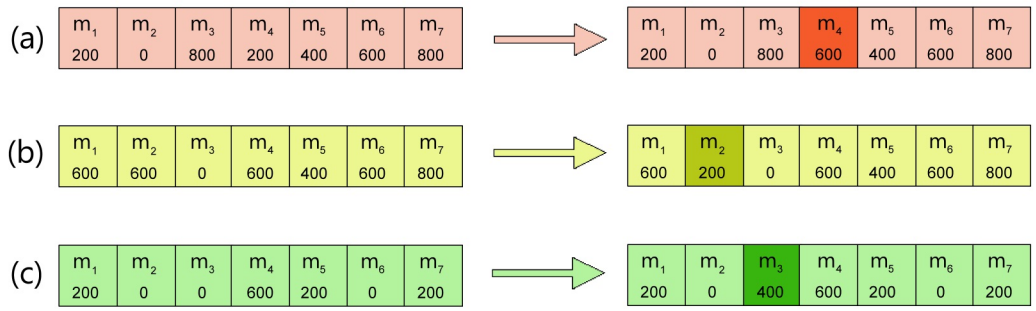

Figure 4.3: Illustration of mutation applied to child schedules. (a), (b) and (c) are the child schedules which mutation is applied with 1/7 mutation probability.

The overall GA for offset scheduling, that is developed in this thesis is shown in Algorithm 6. It follows the general procedure of a GA in Algorithm 5, with our particular definitions of the crossover, mutation, etc.

Algorithm 6 Input: *message set Z, maxIterationNo, populationNo, mutationRatio, crossoverRatio, maxChildrenNo*

Include Standard Offset Schedule to initial population 1

for *i=1 to populationNo-1* 2

  Generate a schedule by applying mutation to Standard Offset Schedule and include to initial population 3

end 4

for each *schedule in the population* 5

  Do the Offset WCRT analysis and check the schedulability of the schedule 6

  if *schedulable* 7

    stop and return the schedulable solution 8

  else 9

    evaluate the fitness of the schedule as sum of the slack of the messages 10

end 11

for *iteration x=1 to maxIterationNo* 12

  Select two parent schedules from the population having best fitness. 13

  Cross over the parent schedules with crossoverRatio to generate new schedules. 14

  Make mutations to new schedules with mutationRatio. 15

  for each *new schedule* 16

    Do the Offset WCRT analysis and check the schedulability of the schedule 17

    if *schedulable* 18

      stop and return the schedulable solution 19

    else 20

      Evaluate the fitness of the schedule as sum of the slack of the messages 21

      Put new schedule into the population 22

  end 23

  Keep the amount of populationNo schedules having best fitness 24

end 25

return the best solution in current population 26

CHAPTER 5

Experimental Evaluation of the Developed Offset Scheduling Algorithms

5.1 Development Environment

The algorithms are implemented in the C++ programming language, and compiled with Microsoft Visual Studio as a Console Application for the Windows operating system. Implementations are built on *Automotive Scheduler C++ API* which is created by Assist. Prof. Dr. Klaus Schmidt [10]. As a C++ implementation, the natural application interface of the *Automotive Scheduler* is given by C++ class and function declarations. The *Automotive Scheduler* is organized in four components, namely *CAN Tools* for the simulation of CAN networks, *FlexRay Tools*, *Automotive Tools* and *Basic Functions* for providing basic functionality.

5.2 Existing Functionality of the Automotive Scheduling Software

In the implementations of the thesis, mainly `Message` and `CANScheduler` classes of the `Automotive Scheduler` are used. `Message` class is used to define the CAN messages of the network. This message model describes the properties of a message that are relevant for CAN scheduling. ID, station number, period, length, offset etc. properties of the message can be set and be accessed by the respective member functions.

The `CANScheduler` class takes a set of CAN messages and either analyses if a given priority assignment is schedulable or tries to find a schedulable priority assignment. The existing Scheduler is limited to the classical case where offsets are not considered. `Analyze` function of The `CANScheduler` class analyses whether the given message set is schedulable or not by

using the algorithm described in 2.2.2. The `AnalyzeMessage` function is called for each message in order to analyze the schedulability of a specific message.

5.3 Implemented Functions in the Scope of the Thesis

The offset scheduling algorithms developed in this thesis and the algorithm which performs worst-case response time analysis for CAN messages with offsets are implemented as the functions of the `CANScheduler` class.

Standard Offset Scheduling Algorithm (SOSA) explained in Section 3.2 is implemented in function `offsetSchedule`. Input of the function is a *message vector*. Even though offsets are already assigned to any message, this function independently assigns offset for each message in the vector. Messages are grouped according to their nodes and then *SOSA* is applied to each group independently.

`offsetWCRTanalyze` function is the implementation of the algorithm described in Section 3.3. This function takes a `message vector` as input whereby, it is assumed that offsets are pre-assigned, for example according to the *SOSA*. It analyzes the worst-case response times of each message and determines whether the message is schedulable or not. If all the messages are schedulable, then function returns `true`; otherwise if any message is unschedulable then it returns `false`. In addition, the function can return the respective worst-case response times.

`offsetSchedulingGA` function finds an offset schedule as performing a genetic algorithm explained in the Section 4.3. Input of the function is a `message vector`. `offsetSchedulingGA` searches for a schedulable solution at most for a predefined number of iterations. It stops when it finds a schedulable solution and returns it; otherwise it returns the best solution in the population.

The proposed solution to find an offset schedule which is described in Section 4.2 is implemented in `offsetLNSA` function. Initially, offsets are assigned to the messages with the *SOSA* because the *Standard Offset Schedule* is used as a basis and then do the WCRT analysis to successively verify schedulability and then change the schedule by keeping the best solution. Finally, it returns the best solution it can find.

5.4 Description of NETCARBENCH

In order to test the scheduling algorithms, we need to generate message sets. We generate the message sets by using NETCARBENCH tools that are developed to be used in the design of in-vehicle communication networks [16].

NETCARBENCH allows users to define the parameters of the generated message sets in a configuration [16]. The outcomes of NETCARBENCH are satisfactorily close to the input specifications. User can define the characteristics of messages and the network: such as load and bandwidth of the network, number of nodes in the network, periods and length of the message with defined weight, load of the stations, etc [11]. NETCARBENCH also assigns the IDs of the messages randomly among the interval defined in the configuration file.

5.5 Generated Message Sets

In order to test the performances of the algorithms, we define five different classes of message sets to be generated by NETCARBENCH. We name these classes as *ClassA*, *ClassB*, *ClassC*, *classD* and *ClassE*. Main difference between these classes is the *load* configuration of the network. *ClassA* has the minimum load configuration, while *classE* has the maximum.

Main property of *ClassA* is to have very low utilization so that all combinations of offsets, all search space, can be explored in a reasonable time. Periods used in *ClassA* are 6, 8 and 10 ms. Load of the network is defined as 17%-18% and number of the ECUs is 3. The *granularity* configuration is set to 2 ms. Bandwidth of the network is 200 Kb/s. According to the conducted experiments, NETCARBENCH generates average 10,87 messages per a test with this configuration.

All configurations of *ClassB*, *ClassC*, *ClassD* and *ClassE* are same except for the *load* configuration of the network. These classes are created by using the message properties of *Society of Automotive Engineers (SAE)* benchmark [17]. The SAE report describes a total of 53 messages assigned to seven different subsystem in a prototype car. It provides a good example to illustrate the application of CAN. Periods of the messages defined in this benchmark are 5, 10, 20, 50, 100 and 1000 ms. The *granularity* configurations of these classes are set to 2 ms. Number of the ECUs is 5-7 and bandwidth of the network is 500 Kb/s. Weight of the periods

and weight of the message lengths are assigned according to SAE benchmark data.

load of *ClassB* is set to 25%-30%. Average number of messages generated with this class is 30.71. *ClassB* has low utilization and different algorithms can easily find a schedulable solution. *load* configurations of *ClassC*, *ClassD* and *ClassE* are set to 45%-50%, 65%-70% and 75%-80% in order. For each test, NETCARBENCH generates some number of messages in order to satisfy the defined parameters. With these load configurations, NETCARBENCH generates average of 54.42, 77.14 and 88.56 messages per a test for *ClassC*, *ClassD* and *ClassE* respectively.

Table 5.1: Configuration of the Message Sets Used in the Experiments

Network	Load (%)	# ECUs	Average # of Messages	Bandwidth	Frame Periods
Class A	17-18	3	10.87	200 kbps	6 ms -10 ms
Class B	25-30	5-7	30.71	500 kbps	5 ms - 1 s
Class C	45-50	5-7	54.42	500 kbps	5 ms - 1 s
Class D	65-70	5-7	77.14	500 kbps	5 ms - 1 s
Class E	75-80	5-7	88.56	500 kbps	5 ms - 1 s

5.6 Comparison and Discussion

In order to test the performances of the scheduling algorithms described in the thesis, message sets are generated by using NETCARBENCH for each message set class defined in Section 5.5. For each message set, all scheduling algorithms are applied and then their performances are evaluated in terms of schedulability, number of unscheduled messages, average of average slacks and run time. Message sets are generated for each class until every simulation results of every algorithms are stabilized in other words until 90% confidence interval is achieved [18]. For each class, between 500-1000 message sets are generated. In this section, we compare the results of different algorithms with respect to schedulability ratio, average number of unschedulable messages, average of average slacks and average run-time.

5.6.1 Test Results Obtained for Class A

Although the load configuration of this class is low, period configurations of this class are chosen to be so small such that it is difficult to find a schedulable solution. The reason why

load of the bus and lengths of the periods are kept low is that the whole search space can be explored in a reasonable time, in order to obtain optimum schedule having the maximum average slack with schedulability constraint. Thus, the performance of the scheduling algorithms can be compared with the optimum schedule.

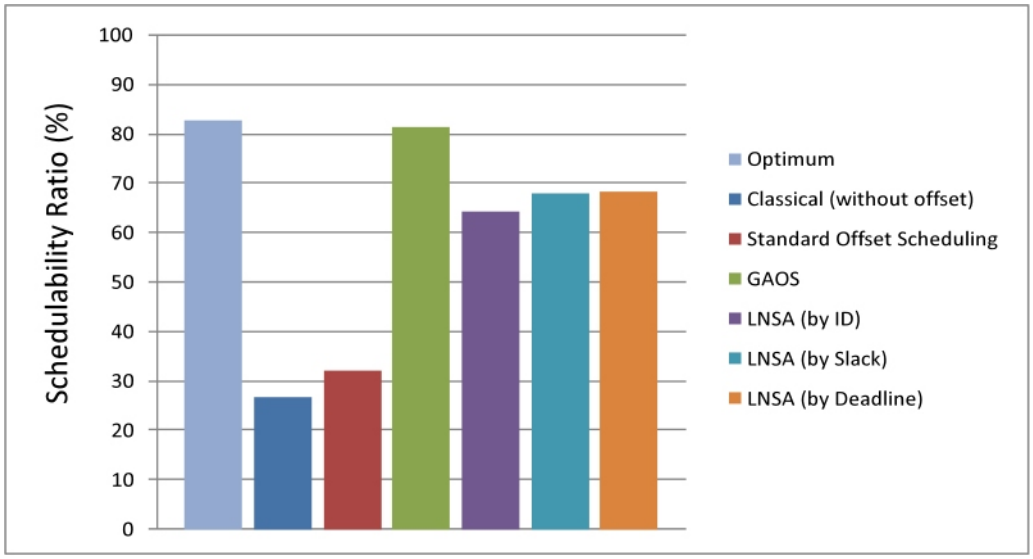

Figure 5.1: Schedulability Performance of the Algorithms for Class A

Since the most important performance metric is the schedulability of the system, plots for the schedulability ratio give major idea about the performances of the algorithms. Figure 5.1 indicates that *GAOS* produces good results when the number of message is small, and performance of *GAOS* is very close to the optimum solution in terms of schedulability which means, almost whenever a feasible schedule exists, it is found by *GAOS*.

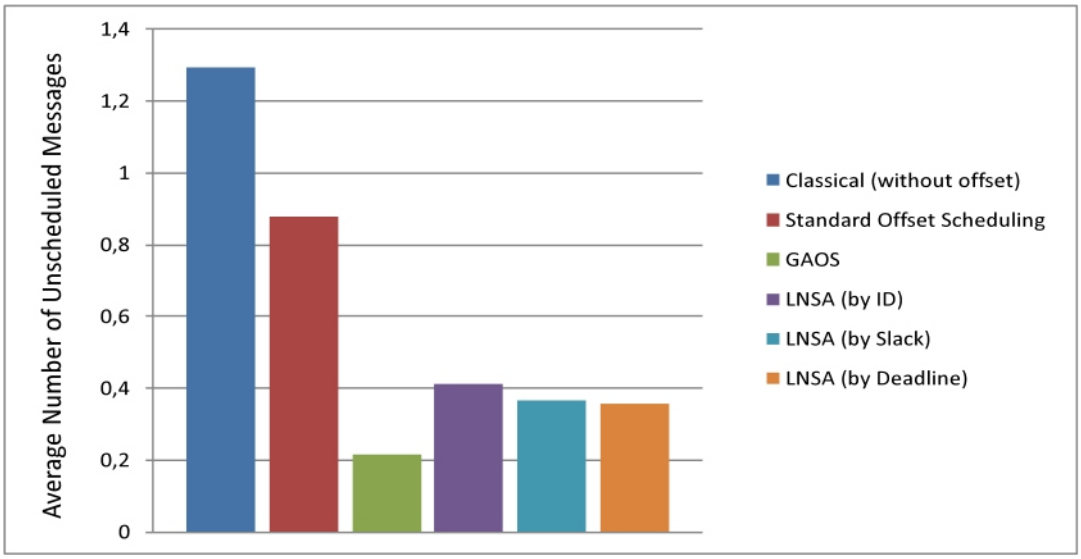

Figure 5.2: Average Number of Unscheduled Messages for Class A

Average number of unschedulable messages is obtained by averaging the number of messages which are not made schedulable by the corresponding scheduling algorithm over the number of tests conducted for the corresponding class. Hence, this number gives parallel idea that how much an algorithm is successful to make messages schedulable. As can be seen from Figure 5.1 and Figure 5.2, number of unscheduled messages per algorithm is inversely proportional to schedulability performance of that algorithm.

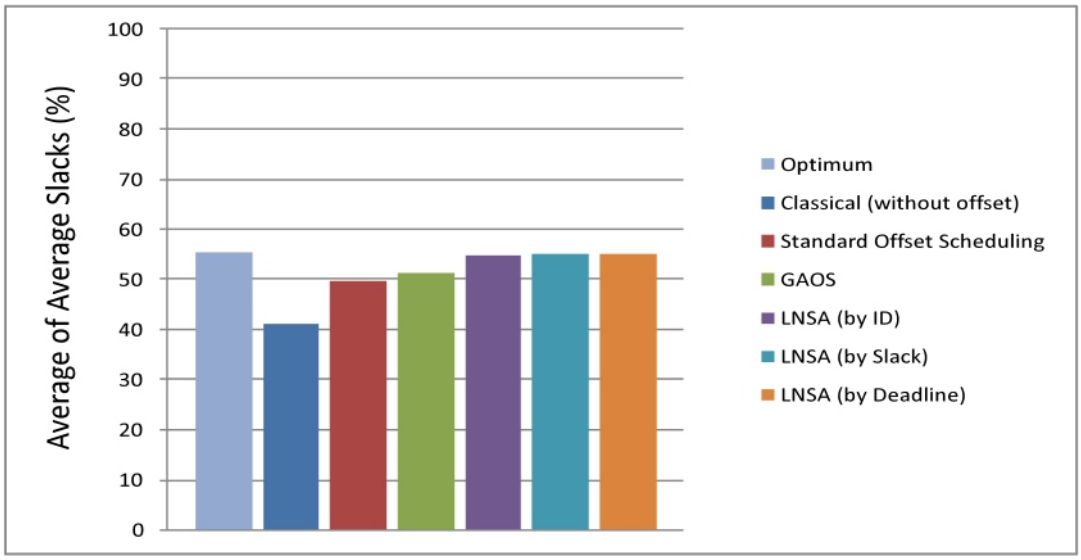

Figure 5.3: Average Slacks for Class A

As can be seen from Figure 5.3, although schedulability performance of the algorithms are

very different, average slacks of different algorithms are not very different. So, it can be concluded that to have a good distribution of the available slack among the different messages is very important.

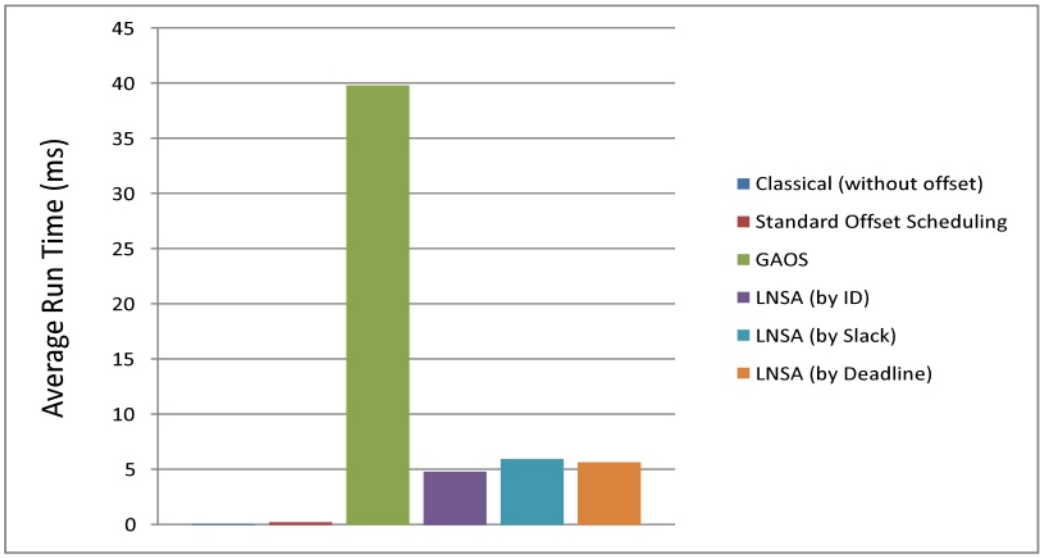

Figure 5.4: Average Run-Time for Class A

Run times of the algorithms developed in this thesis are always higher than *Classical* and *SOSA* because the complexity of the developed algorithms is higher when compared to *Classical* and *SOSA* because of the integration of the WCRT analysis in the schedule computation. Run times of LNSAs of different sorting strategies are always close to each other for any configuration because LNSA tries the same number of schedules and hence performs the same number of WCRT analysis for the same message set. It can also be seen that *Genetic Algorithm* runs longer than *LNSA*s for *Class A*. However, average number of WCRT analysis of *GA* is 19 while it is 35 for *LNSA*. This indicates that *GA* also spends time to perform mutation, crossover, etc. which influences the average run time for a schedule when compared to *LNSA*.

5.6.2 Test Results Obtained for Class B

Class B is configured such that the bus utilization is adjusted low, and so the scheduling algorithms can easily find a schedulable solution.

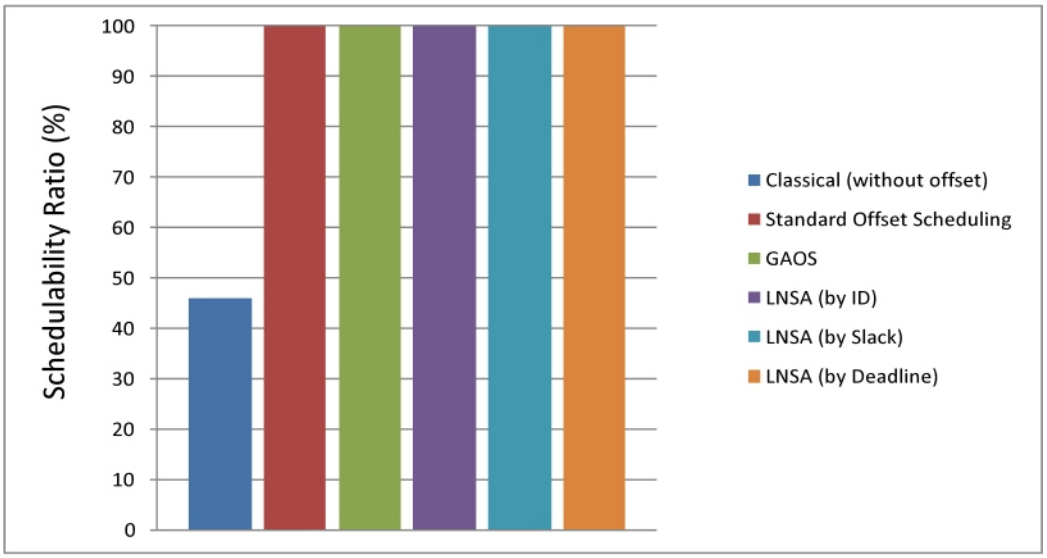

Figure 5.5: Schedulability Performance of the Algorithms for Class B

As can be seen from the results above, it is confirmed that offset scheduling algorithms have importantly higher performance when compared to *Classical Scheduling* in which offsets are not considered. For the configuration of this class, any offset scheduling algorithm finds a schedulable solution.

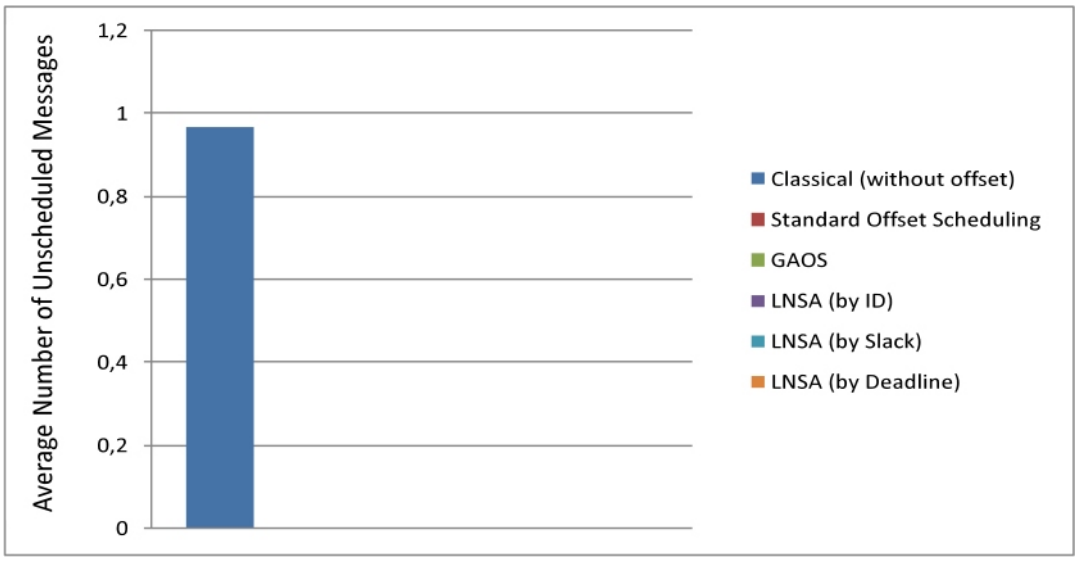

Figure 5.6: Average Number of Unscheduled Messages for Class B

There are not any unscheduled messages for any offset scheduling algorithms because they always make the messages meet their deadlines at this low traffic load, however; *Classical Scheduling* may still not succeed as can be seen from Figure 5.6.

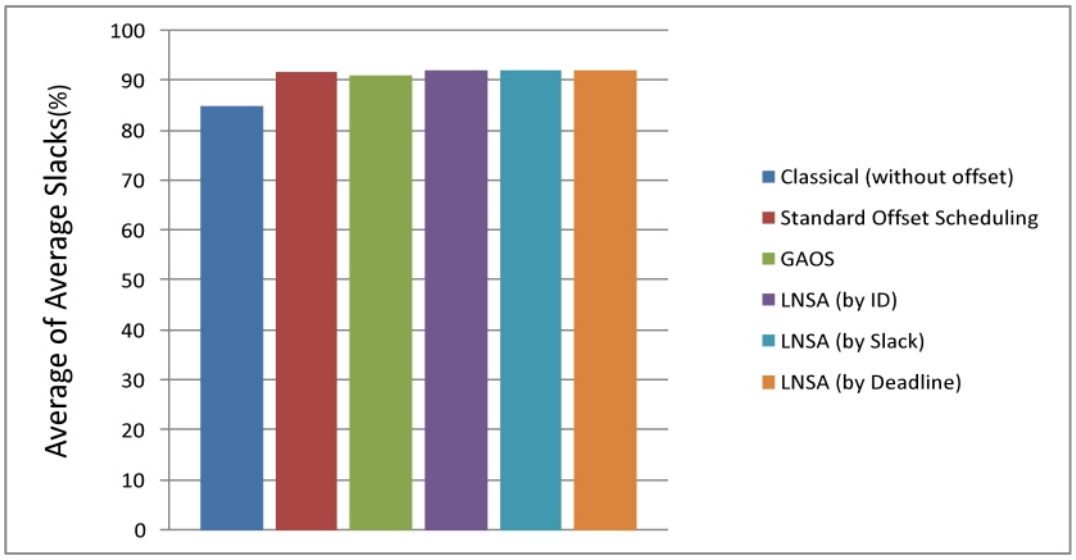

Figure 5.7: Average Slacks for Class B

Figure 5.7 shows that average slacks of the messages are high. This indicates that worst case response times of the messages are low, which is expected because bus utilization of this Class is low.

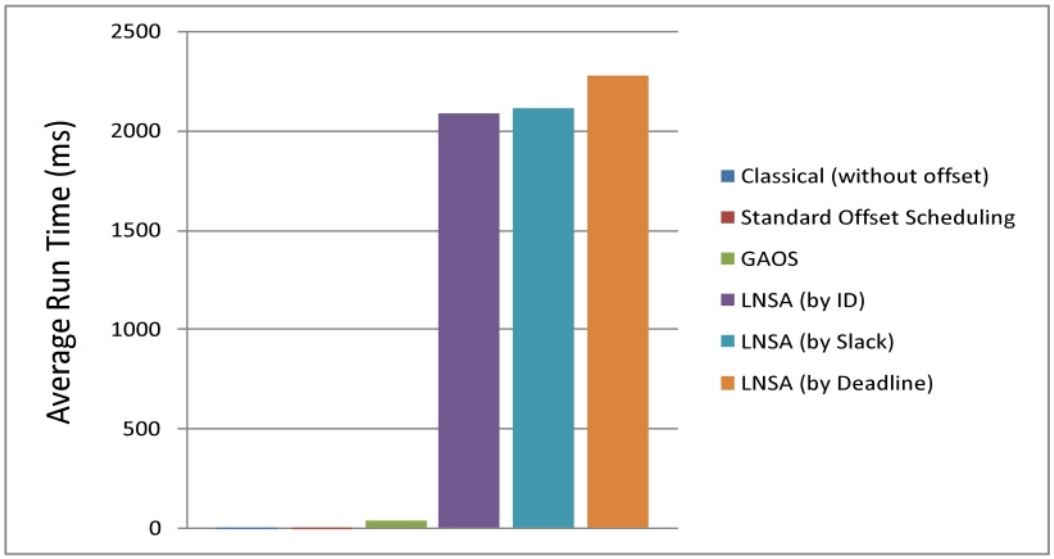

Figure 5.8: Average Run-Time for Class B

For this class, run time of *Genetic Algorithm* is less than *LNSA*s. Average number of WCRT analysis of *GA* is 15 while it is 938 for *LNSA*. The reason is that *GA* uses solution of *Standard Offset Scheduling* to generate initial population, which is already schedulable for this class; *GA* immediately finds a schedulable solution so does not need to make any iterations. It should be noted that *GA* always tries at least 15 schedules since the number of initial population is

set to 15.

5.6.3 Test Results Obtained for Class C

Load of the network increases 20% when compared to *Class B*, so it becomes more difficult to make the messages meet their deadlines for this message set configuration.

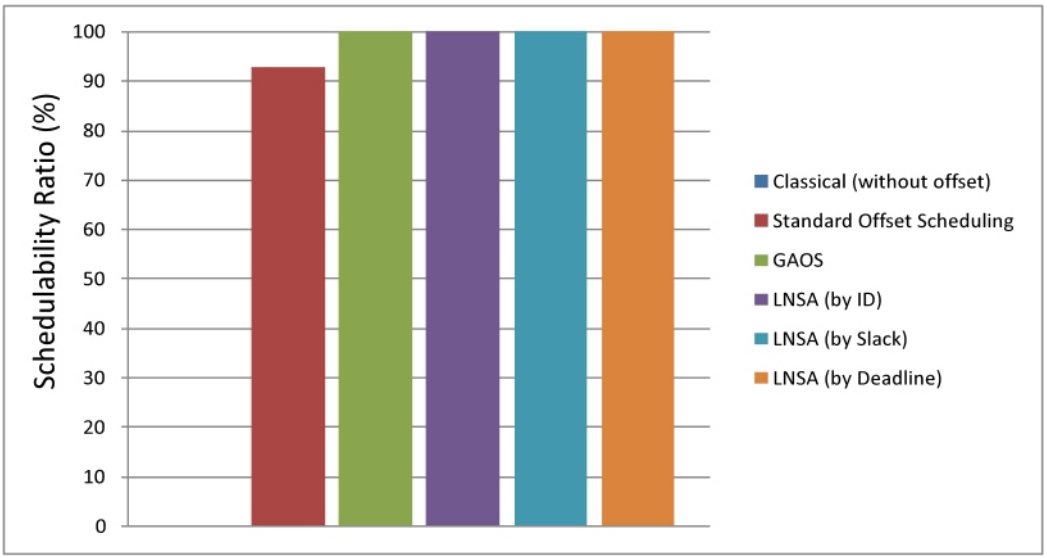

Figure 5.9: Schedulability Performance of the Algorithms for Class C

As can be seen from Figure 5.9, *Classical Scheduling* can not find a schedulable solution at loads of 45%-50%. On the other hand, *SOSA* usually finds a feaible schedule, where developed algorithms always do.

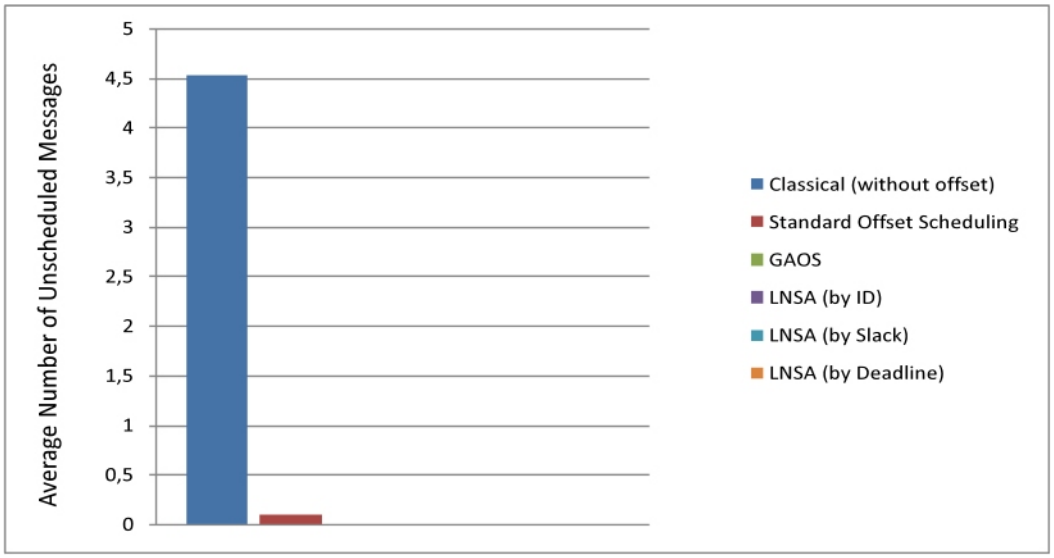

Figure 5.10: Average Number of Unscheduled Messages for Class C

Figure 5.10 also validates that *Classical Scheduling* is weak when compared to offset scheduling in terms of schedulability of the messages and so the systems. Moreover, it can be deduced that developed algorithms always make the messages meet their deadlines for this message set configuration, while *SOSA* does not.

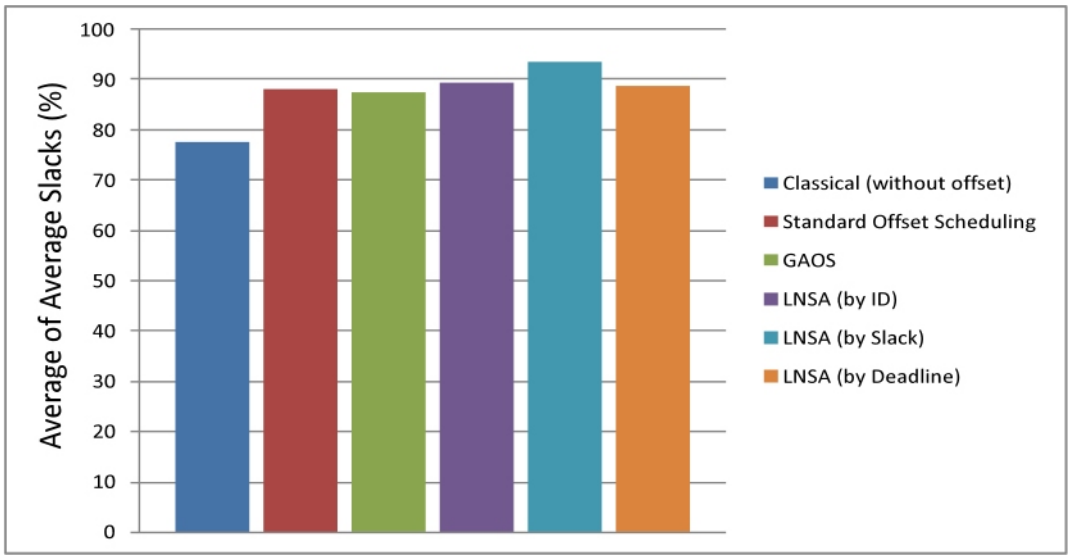

Figure 5.11: Average Slacks for Class C

Figure 5.11 states that the worst case response times of the messages increase as bus utilization increases, which is expected.

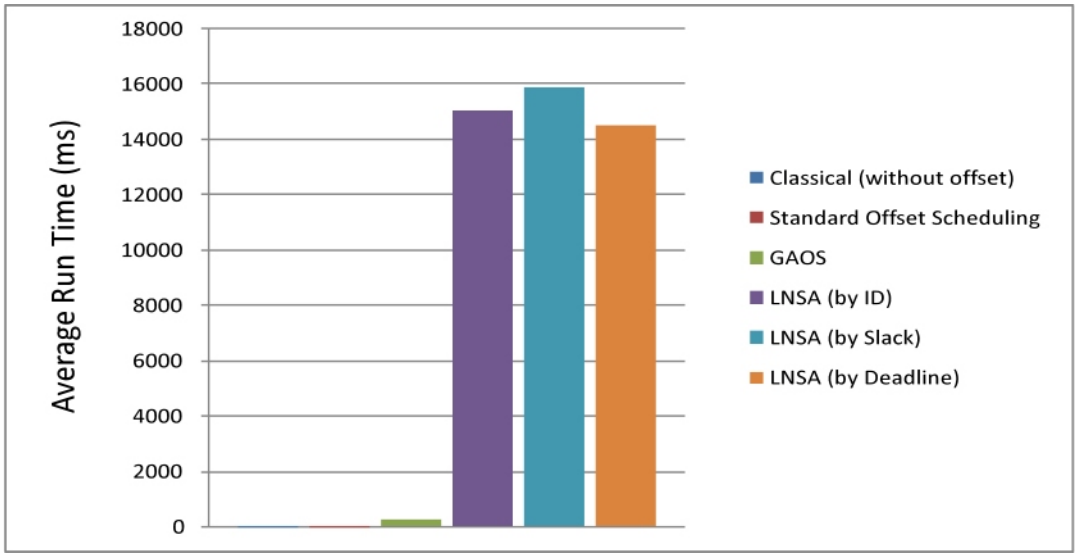

Figure 5.12: Average Run-Time for Class C

Run times of *LNSA*s are comparably high when compared to *GAOS* because *LNSA* explores average 1674 different schedules while *GAOS* does 15 that corresponds to the number of initial population. Average number of schedules that *LNSA* explores is linearly proportional to average number of messages. However, when Figure 5.8 and Figure 5.12 are considered, run times of *LNSA*s are not directly proportional to average number of messages because run time of WCRT analysis also increases as number of messages increases.

5.6.4 Test Results Obtained for Class D

Load of the network is now increased to 65%-70% for *Class D*.

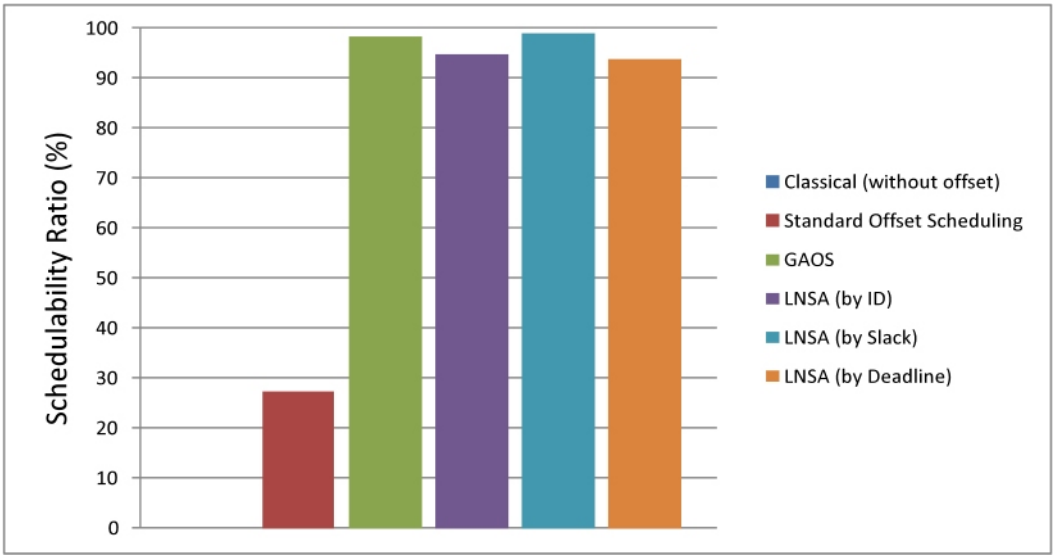

Figure 5.13: Schedulability Performance of the Algorithms for Class D

Even for this class which has configuration of 65%-70% load; developed algorithms can easily find a schedulable solution, while *SOSA* can hardly find a schedulable solution. This already indicates the limitations of *SOSA*.

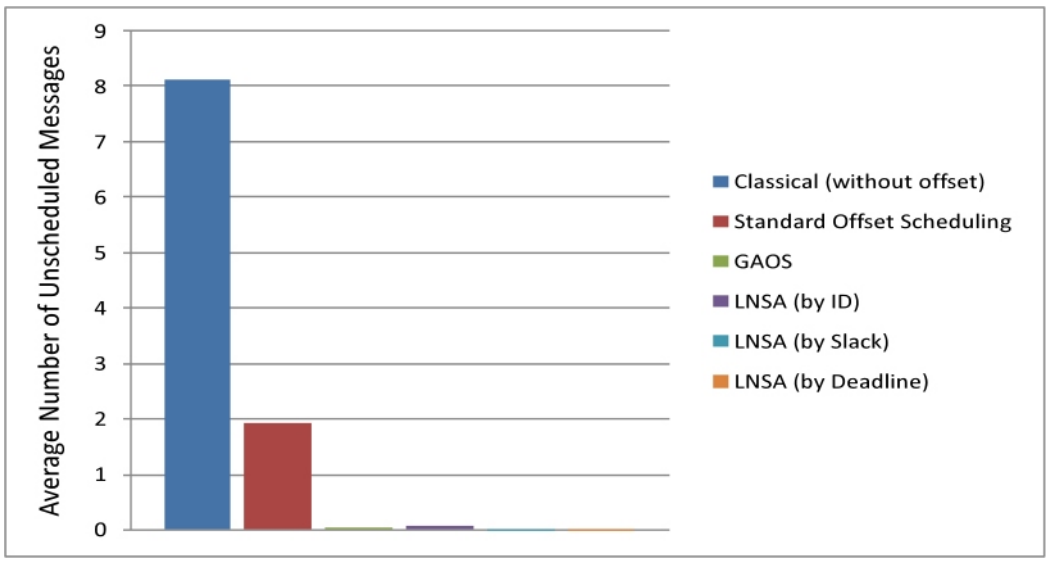

Figure 5.14: Average Number of Unscheduled Messages for Class D

Average number of unscheduled messages are not zero but very low for the developed algorithms, while it is very high for *Classical Scheduling*. This confirms that controlling the

release times of the messages is very beneficial in terms of the worst-case response times.

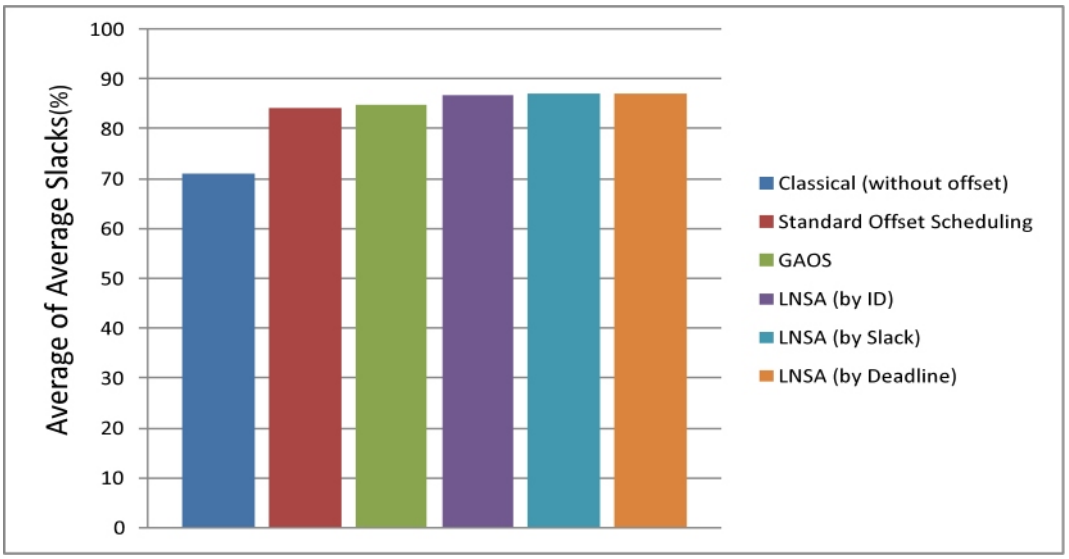

Figure 5.15: Average Slacks for Class D

Average slacks for the different algorithms are still very close to each other, while schedulability performances of the developed algorithms are significantly higher than performance of *SOSA*. Hence, it can be concluded that some messages get large slacks while other messages fail in the *SOSA*.

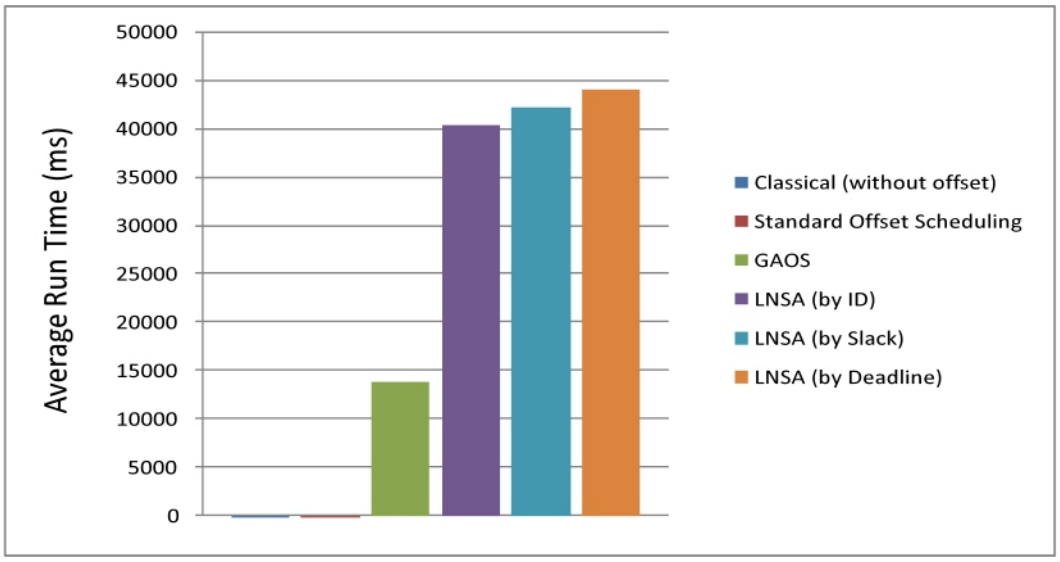

Figure 5.16: Average Run-Time for Class D

For this class, run time of *GAOS* visibly increases because *GAOS* does not terminate in the first iterations, as was observed for the previous traffic classes. Instead, *GAOS* makes average number of 143 iterations until termination.

5.6.5 Test Results Obtained for Class E

Bus utilization, load, of *Class E* is adjusted to 75%-80%, which is very high. For the tests done for *Class E*, *GAOS* is tested with different maximum number of iterations. In previous tests, maximum number of iterations, *maxIterationNo*, is adjusted to 500, which is sufficient for low utuilization. However, this number is low for *GAOS* to explore search space at high utilizations as can be seen from the results below. For the following figures, *maxIterationNo* of *GAOS* is indicated inside the paranthesis after *GAOS*; such as *GAOS(500)* where *maxIterationNo*=500.

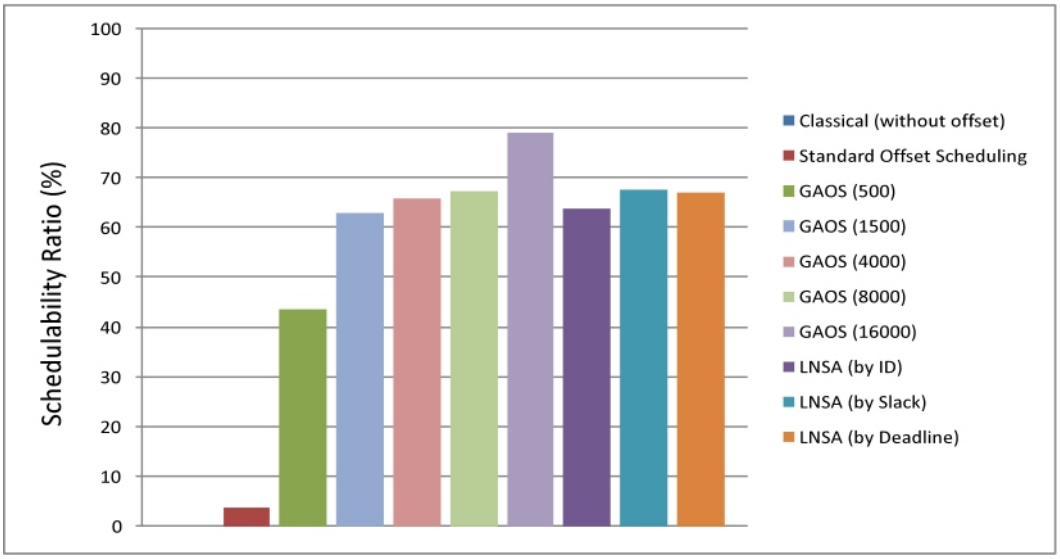

Figure 5.17: Schedulability Performance of the Algorithms for Class E

As can be seen from Figure 5.17, *Standard Offset Scheduling* almost can not find a schedulable solution for this class. *LNSA (by slack)* and *LNSA (by deadline)* can find a schedulable solution most of the time. Another issue to be considered is that when *maxIterationNo* increases, schedulability of *GAOS* increases. This is due to the fact that, *GAOS* has more opportunity to explore the search space if *maxIterationNo* is higher.

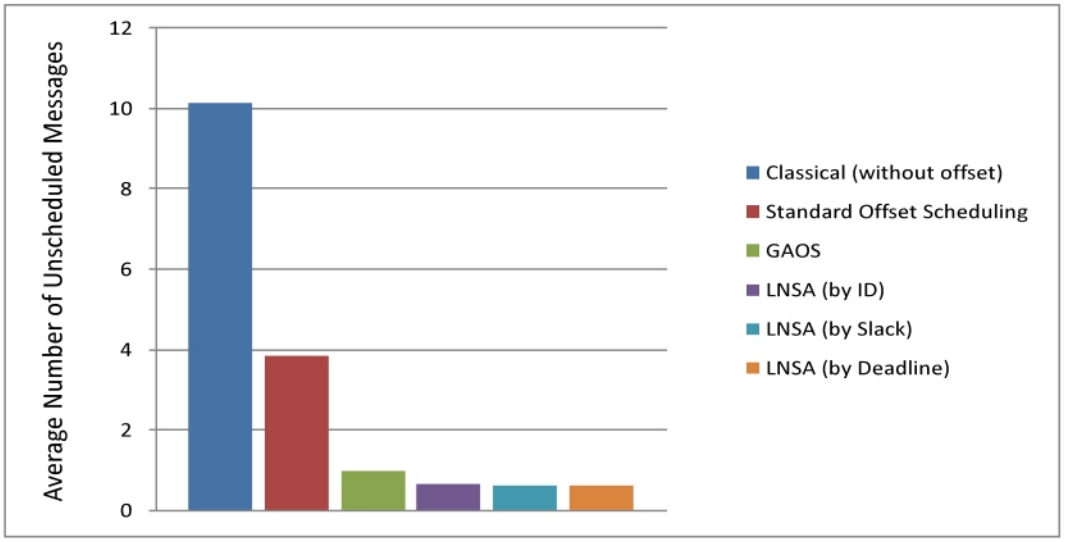

Figure 5.18: Average Number of Unscheduled Messages for Class E

Figure 5.18 confirms the arguments mentioned above about the performances of the algorithms. In particular, number of unschedulable messages for the developed algorithms is much lower than for the existing algorithms.

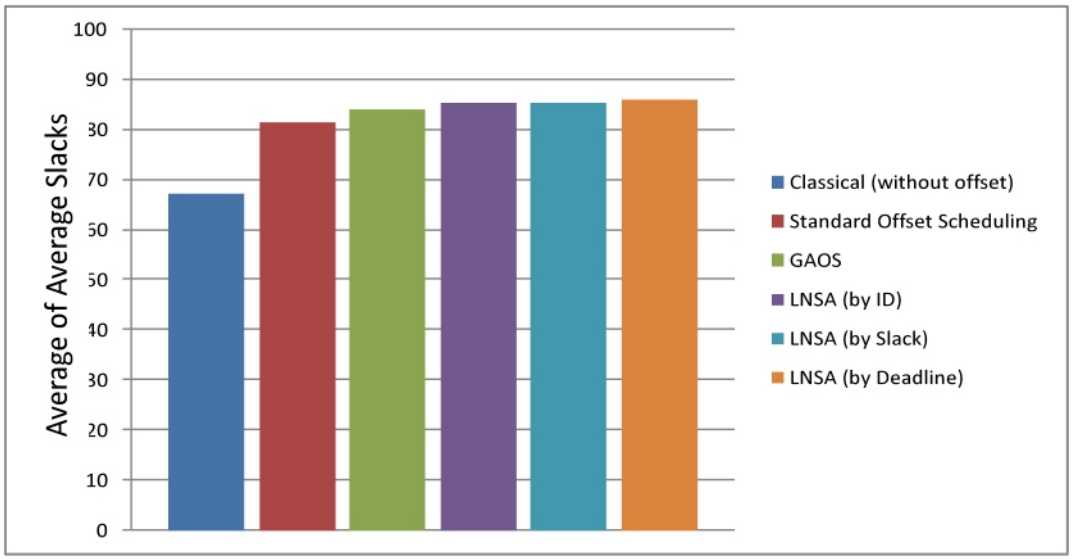

Figure 5.19: Average Slacks for Class E

As can be seen from the Figure 5.19, it is confirmed that to have a good distribution of the available slack among the different messages is very important. Because average slacks of different algorithms are not very different, although schedulability performances of the algorithms are very different.

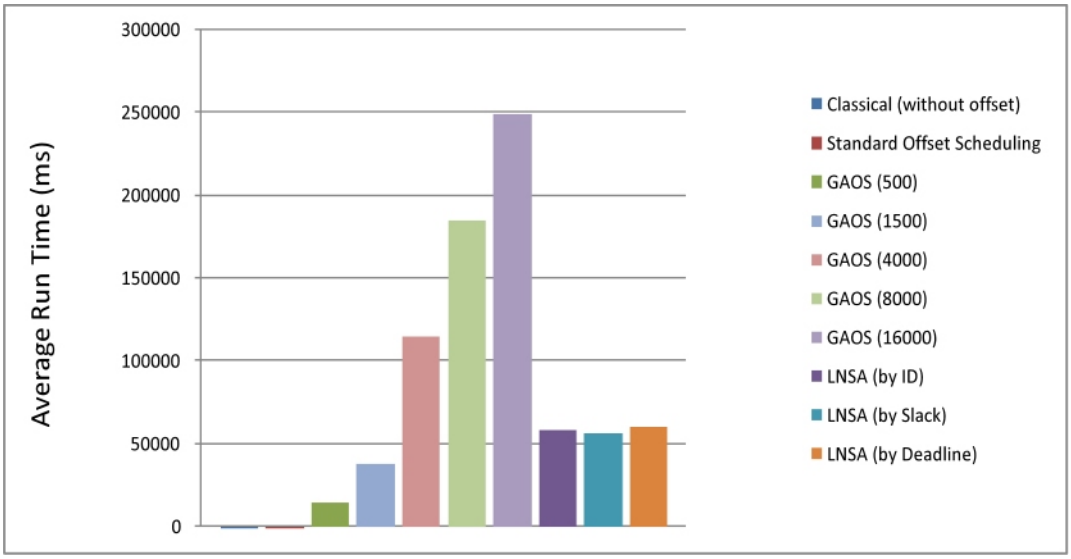

Figure 5.20: Average Run-Time for Class E

Run time of *GAOS* shows a significant increase with the maximum iteration number, if the bus utilization is high. Figure 5.20 shows that run time of *GAOS* increases as the maximum iteration number increases and so passes over run times of *LNSA*s. However, the run time for the developed algorithms is less than 5 minutes which is practically feasible.

5.6.6 Test Results Obtained for GAOS with the Solution of LNSA

For the previous tests, *Standard Offset Schedule* is used to generate initial population to be used in *GAOS*. In this test, it is aimed to evaluate the performances of *GAOS* by using the solution of *LNSA(deadline)*. In this test we name this combination as *LNSA(deadline)+GAOS(16000)*. It is expected to obtain better performances because *LNSA(deadline)* produces better schedules than *SOSA* as shown in previous results. Due to the nature of *Genetic Algorithm*, *GAOS* has more opportunity to find a schedulable solution if it starts evolving the optimization from a better schedule. Network properties used in this network is as defined in *Class E* and maximum iteration number of *GAOS* is adjusted to 16000.

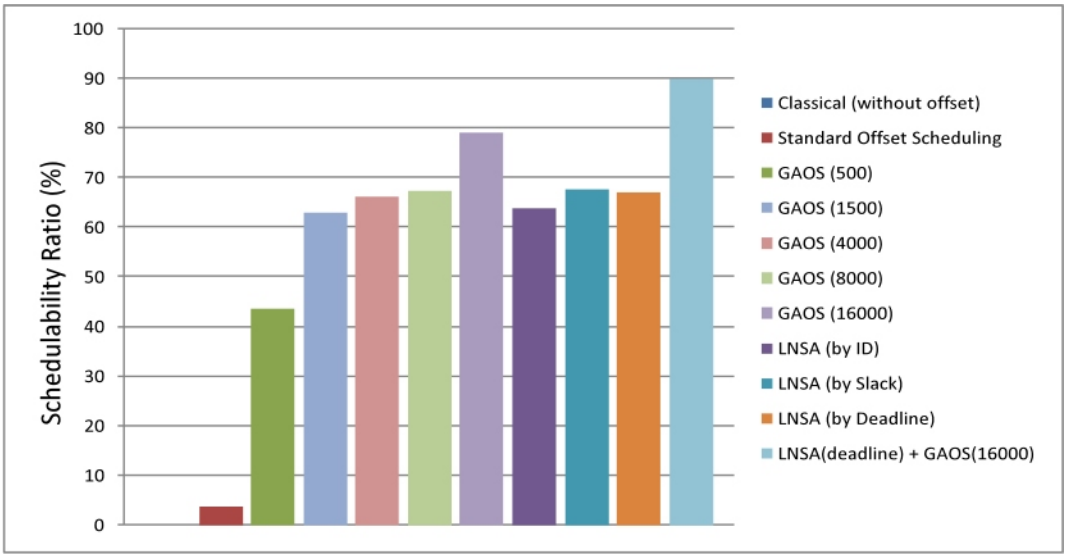

Figure 5.21: Schedulability Performance of the Algorithms for Class E

As can be seen from Figure 5.21, combination of *LNSA* and *GAOS(16000)* produces the best result among the overall experiments for *Class E* conducted in this thesis. Actually, *GAOS(16000)* and *LNSA(deadline)+GAOS(16000)* are the applications of *GAOS* with same parameters; however; schedulability performance of *LNSA(deadline)+GAOS(16000)* is %90, while %79 for *GAOS(16000)* which uses *Standard Offset Schedule* to generate initial population. This indicates that *GAOS* reaches a schedulable solution more successfully if it starts with better initial population.

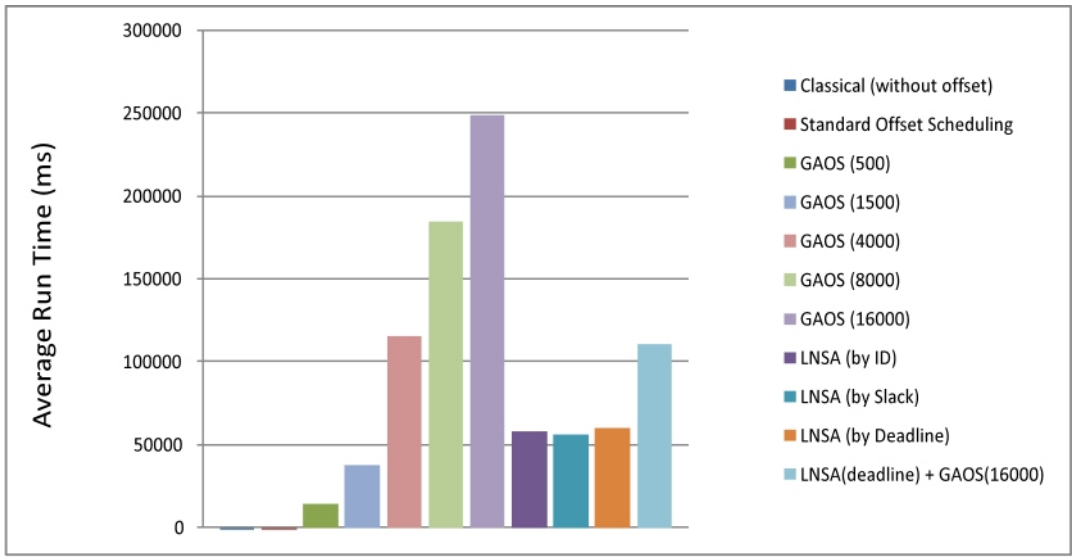

Figure 5.22: Average Run-Time for Class E

Figure 5.22 shows that runtime of *LNSA(deadline)+GAOS(16000)* is about 110 seconds while

it is 250 seconds for *GAOS*. This also confirms that starting with a better initial population improves effectiveness of *GAOS*.

5.6.7 General Observation on the Results

Experimental results confirm that offset scheduling is beneficial in terms of the worst-case response time and so the schedulability. Nicolas Navet et. al. concentrates on WCRT of the lowest priority messages while comparing the performances of *Classical Scheduling* and *Standard Offset Scheduling* [3]. On the contrary, we have compared the different scheduling algorithms especially in terms of schedulability and average slack of the system. However, results obtained in this thesis are parallel to obtained in [3] in sense that both show how the worst-case response times dramatically decrease with offset scheduling. Moreover, the results clearly indicate that *Standard Offset Scheduling* can be used for low utilizations up to 50% and realistic message properties (period,length) as in the SAE set [17].

Experimental results show that the developed algorithms clearly outperform the *SOSA*, especially for high utilizations. In particular, for utilizations between 65%-75%, the developed algorithms find a feasible schedule in more than 95% of the test case, whereas the *SOSA* is succesful in only 27% of the test cases. The improvement is even higher for utilizations between 75%-80%. Here, the developed algorihtms find feasible solutions in about 90% of the test cases, whereas the *SOSA* finds a solution in only 4% of the test cases. It further has to be noted that in all of the experiments, the run time for the developed algorithms is less than 5 minutes, and hence suitable for practical applications.

Considering the developed algorithms, it is observed that *GAOS* is very suitable for small messages sets and low utilizations because of its small run time. The run time of the *GAOS* increases considerably for large utilizations about 90% if a good schedulability performance shall be achieved. The performance of *GAOS* increases if it starts with better initial population which is expected due to the nature of Genetic Algorithm. Among the *LNSA* algorithms, the best schedulability peformance is achieved when sorting by slacks or sorting by deadline. Both sorting strategies give similar results.

In this thesis, schedulability performance of GAOS is evaluated by adjusting its parameters to the values as mentioned in Section 4.3.2. However, the parameters used in *GAOS* such as

population size, maximum number of child schedules, number of parent schedules, mutation probability and crossover probability can be tested with different values and better performances than obtained in this thesis may be obtained.

The disadvantages of the offset scheduling algorithms developed in this thesis are that their complexities and run times considerably high when compared to SOSA. However; as discussed in this section, the developed algorithms are suitable for practical applications.

CHAPTER 6

CONCLUSION

Technological improvements in the automotive industry increase the demands for usage of CAN bus more efficiently. Currently, only about 40% of the bus is utilized in order to make the messages meet their deadlines at worst-case. In order to overcome this problem, Nicolas Navet et. al. proposed a low-complexity and high performance solution [8, 3]. Main property of this solution is to control the release times of the messages, in which the first instance of a message is released with offset. Main purpose of this algorithm is to distribute the messages as uniformly as possible over time, in order to avoid synchronous releases. As shown in this thesis, this algorithm can find feasible schedules for loads of about 50%, but not higher loads. The main reason for this deficiency is that the algorithm in [8, 3] does not consider the dependencies among messages that are released from different nodes by not including any WCRT analysis in the schedule computation.

It is stated that search space of offset scheduling is too large to compute all of them in a reasonable time for realistic message sets. Consequently, offset scheduling problem is considered as an optimization problem which maximizes overall slack time of the system with schedulability constraint. Two different algorithms are developed to solve this optimization problem. The first one is *Local Neighborhood Search Algorithm* which always changes a single offset and keeps the ”best” solution. The second one is *Genetic Algorithm for Offset Scheduling* which is inspired by natural evolution and keeps a set of ”fittest” solutions and evolve the optimization from there.

In this thesis, in addition to developed algorithms which are *LNSA* and *GAOS*, *SOSA* and worst-case response time analysis for offset scheduling are implemented in C++ by using *Automotive Simulator C++ API* [10]. In order to test the performances of the algorithms,

messages are generated with NETCARBENCH which is used in the design of in-vehicle communication networks. In the experiments, it is cared that 90% of confidence interval is achieved for each scheduling result.

In this thesis, five different experiments are conducted for five different network configuration in order to evaluate the performances of the scheduling algorithms at different situations. First conclusion deduced from the experimental results is the confirmation of the performance increase with offset scheduling as claimed in [3]. Experimental results also show that the developed algorithms definitely produces better results than *SOSA*. The improvement can clearly be seen especially at high loads, in which developed algorithms can find a feasible schedule about 90% of the test cases while *SOSA* is successful in only 4% of the cases. It should be noted that run times of the developed algorithms are bounded by 5 minutes in our experiments which are conducted by a standard personal computer. Hence, the developed algorithms in this thesis can be used in practical applications.

REFERENCES

[1] I. Standard-11898. *Road vehicles-interchange of digital information - Controller Area Network (CAN) for high-speed communication*. International Standards Organisation (ISO), 1993.

[2] I. Standard-11898. *FlexRay Communication System, Protocol Specification, Version 2.0.*. http://www.flexray.com, 2004.

[3] M. Grenier, L. Havet, and N. Navet. *Pushing the limits of CAN - scheduling frames with offsets provides a major performance boost*. Automotive Embedded Systems Handbook, Industrial Information Theory, vol. 5, 2009.

[4] R. I. Davis, A. Burns, R. J. Bril, and J. J. Lukkien. *Controller area network (CAN) schedulability analysis: Refuted, revisited and revised*. Real-Time Systems, vol. 35, no. 3, pp. 239–272, April 2007.

[5] L. George, N. Rivierre, and M. Spuri. *Pre-emptive and non-pre-emptive real-time uni-processor scheduling*. Technical Report 2966, Institut National de Recherche et Informatique et en Automatique (INRIA), France, 1996.

[6] K. W. Schmidt. *Novel Controller Area Network Scheduling Techniques for Practical Applications*. submitted to Industrial Informatics, IEEE Transactions, 2011.

[7] K. Tindell, A. Burns, and A. Wellings. *Calculating controller area network (CAN) message response times*. Control Engineering Practice, vol. 3, pp. 1163–1169, 1995.

[8] M. Grenier, L. Havet, and N. Navet. *Scheduling messages with offsets on Controller Area Network - a major performance boost*. European Congress on Embedded Real Time Software, Toulouse, France, 2008.

[9] L. Du, G. Xu. *Worst Case Response Time Analysis for CAN Messages with Offsets*. Vehicular Electronics and Safety, IEEE International Conference on, pp. 41–45, 2009.

[10] K. W. Schmidt. *Automotive Simulator C++ API*. 2007.

[11] C. Braun, L. Havet, N. Navet. *NETCARBENCH v2.2 User Manual* . August 31, 2011.

[12] P. C. Richardson, X. Weidgong, S. Mohammad. *Performance analysis of a real-time control network test bed in a linux-based system with sporadic message arrivals*. IEEE Transactions on Industrial Informatics, vol. 2, no. 4, pp. 231–241, 2006.

[13] R. I. Davis, A. Burns, R. J. Bril, and J. J. Lukkien. *Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised*. Real-Time Systems, vol. 39, pp. 239–272, 2007.

[14] H. Zeng, M. Di Natale, P. Giusto. A. Sangiovanni-Vincentelli. *Stochastic analysis of can-based real-time automotive systems*. IEEE Transactions on Industrial Informatics, vol. 5, no. 4, pp. 388–401, 2009.

[15] J. Lehoczky, *Fixed priority scheduling of periodic task sets with arbitrary deadlines*. 11th IEEE Real-time Systems Symposium, pp. 201–209, 1990.

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

[17] *Class C Application Requirement Considerations*. Society for Automotive Engineers, J2056/1, 1993.

[18] Semih Bilgen. *Confidence of Simulation Results*. unpublished lecture note, METU, 1986.