Master Thesis Report

SIMULATION-BASED END-TO-END LATENCY ANALYSIS OF ADAS SYSTEMS

ISHU GOEL

Integrated Vehicle Safety Department, TNO Automotive

Faculty of Electrical Engineering, Mathematics and Computer Science
SIMULATION-BASED END-TO-END LATENCY ANALYSIS OF ADAS SYSTEMS

THESIS

submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

EMBEDDED SYSTEMS

by

ISHU GOEL
born in HAPUR, INDIA

Computer Engineering
Department of Electrical Engineering
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
SIMULATION-BASED END-TO-END LATENCY ANALYSIS OF ADAS SYSTEMS

by ISHU GOEL

Abstract

Many Advanced Driver Assistant Systems functionalities were introduced to boost vehicle safety, efficiency and comfort. Implementing these functionalities such that all the functional and temporal requirements are satisfied is challenging. This thesis is concerned with the temporal requirements. More precisely, it aims to identify and implement a methodology to estimate the end-to-end latency distribution of various signals paths in ADAS, with the aim of verifying the compliance of latency requirements. The methodology is validated by testing it on a hardware set-up that mimics the behaviour of an existing adaptive cruise control system at TNO. This system is equipped with two different communication buses and multiple ECU’s with different temporal behaviours.

A simulation-based method is used to address the problem. The simulation models of the individual components were first made and then combined to make the model of the complete system. The simulation model uses the input-output latency of the individual components, without any interference from other components, from the hardware model and computes the complex interactions between and within the components. The results of the experiments performed on both simulation and hardware models confirm that a simulation-based method could be used to determine end-to-end latency in a distributed automotive systems.

Laboratory : Computer Engineering
Codenumber : CE-MS-2018-26

Committee Members :

Advisor: Dr. Ir. Arjan van Genderen, CE, TU Delft
Chairperson: Dr. Ir. Zaid Al-Ars, CE, TU Delft
Member: Dr. Ir. Marc Geilen, EE, TU Eindhoven
Member: Dr. Ir. Arturo Tejada Ruiz, IVS, TNO Helmond
Dedicated to my family and friends
# Contents

List of Figures viii  
List of Tables ix  
List of Acronyms xii  
Acknowledgements xiii  

1 Introduction 1  
1.1 Motivation ........................................ 1  
1.2 Problem statement and thesis goals .................. 2  
1.3 Thesis outline .................................. 3  

2 Fundamentals 5  
2.1 Timing metrics ................................... 5  
2.2 ADAS ............................................. 6  
2.3 Challenges of timing analysis in ADAS systems .... 9  

3 Existing methodologies for timing analysis of automotive systems 11  
3.1 Analytical methods .................................. 11  
3.2 Simulation-based methods .......................... 14  
3.3 Comparing simulation-based methods with analytical methods 16  

4 Framework for timing analysis of heterogeneous in-vehicle networks 17  
4.1 Approach ........................................ 17  
4.2 Test set-up ....................................... 23  
4.2.1 CAN setup .................................... 23  
4.2.2 Gateway and Ethernet set-up .................... 24  
4.2.3 World Model ECU implementation ............... 24  
4.3 Simulation model .................................. 26  
4.3.1 Simulation tool .................................. 26  
4.3.2 CAN POOSL model ............................. 27  
4.3.3 Ethernet POOSL model ......................... 29  
4.3.4 Gateway POOSL model ......................... 30  
4.3.5 World model ECU POOSL model ................. 31  

5 Experiments and Results 33  
5.1 Test Plan ......................................... 33  
5.2 Measurement of the latency of the components .... 35  
5.2.1 Tests on CAN model ............................ 35
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>CACC-equipped trucks.</td>
<td>6</td>
</tr>
<tr>
<td>2.2</td>
<td>CACC architecture and message flow in a CACC architecture.</td>
<td>7</td>
</tr>
<tr>
<td>2.3</td>
<td>Chart of maximum end-to-end latency versus time headway for string stable behaviour in CACC, taken from [1].</td>
<td>9</td>
</tr>
<tr>
<td>3.1</td>
<td>Dataflow model of an automotive system and its end-to-end latency, taken from [2]</td>
<td>13</td>
</tr>
<tr>
<td>3.2</td>
<td>CDF of end-to-end delay of transmission of control data and camera data, taken from [3]</td>
<td>15</td>
</tr>
<tr>
<td>3.3</td>
<td>CDF of IAT of transmission of control data and camera data, taken from [3]</td>
<td>15</td>
</tr>
<tr>
<td>4.1</td>
<td>A brief description of the approach taken in this thesis.</td>
<td>17</td>
</tr>
<tr>
<td>4.2</td>
<td>CACC architecture for a single vehicle. The selected part of the architecture is marked by red box.</td>
<td>18</td>
</tr>
<tr>
<td>4.3</td>
<td>Standard CAN data frame, taken from [4]</td>
<td>19</td>
</tr>
<tr>
<td>4.4</td>
<td>Standard IEEE 802.3 Ethernet frame where SFD stands for Start of Frame Delimiter and FCS stands for Frame Check Sequence, taken from [5]</td>
<td>20</td>
</tr>
<tr>
<td>4.5</td>
<td>The functioning of the WM ECU.</td>
<td>21</td>
</tr>
<tr>
<td>4.6</td>
<td>Message flow inside World model ECU.</td>
<td>22</td>
</tr>
<tr>
<td>4.7</td>
<td>Selected system. WM ECU denotes World Model ECU.</td>
<td>22</td>
</tr>
<tr>
<td>4.8</td>
<td>Test set-up.</td>
<td>23</td>
</tr>
<tr>
<td>4.9</td>
<td>World model ECU algorithm flowchart.</td>
<td>25</td>
</tr>
<tr>
<td>4.10</td>
<td>Simulation model.</td>
<td>26</td>
</tr>
<tr>
<td>4.11</td>
<td>Composite structure diagram of the CAN POOSL model.</td>
<td>27</td>
</tr>
<tr>
<td>4.12</td>
<td>Working of the ECU models connected to the CAN bus model.</td>
<td>28</td>
</tr>
<tr>
<td>4.13</td>
<td>CAN bus arbitration code.</td>
<td>28</td>
</tr>
<tr>
<td>4.14</td>
<td>Ethernet bus arbitration code.</td>
<td>30</td>
</tr>
<tr>
<td>4.15</td>
<td>POOSL model showing CAN and Ethernet bus connected through gateway.</td>
<td>31</td>
</tr>
<tr>
<td>4.16</td>
<td>World model POOSL code.</td>
<td>31</td>
</tr>
<tr>
<td>5.1</td>
<td>CAN message latency at different baud rates.</td>
<td>36</td>
</tr>
<tr>
<td>5.2</td>
<td>CAN message latency at different data sizes.</td>
<td>37</td>
</tr>
<tr>
<td>5.3</td>
<td>Frequency distributions of latency of CAN messages (left) and jitter in the period of generation of CAN message (right) from a single-node CAN transmitter in the test set-up.</td>
<td>37</td>
</tr>
<tr>
<td>5.4</td>
<td>Frequency distribution of CAN message latencies generated from normal (left) and discrete (right) distribution of POOSL model.</td>
<td>39</td>
</tr>
<tr>
<td>5.5</td>
<td>Ethernet message latency frequency distribution of the test set-up.</td>
<td>40</td>
</tr>
<tr>
<td>5.6</td>
<td>Test set-up depicting paths used for measuring conversion time inside the gateway.</td>
<td>40</td>
</tr>
</tbody>
</table>
5.7 Frequency distribution of message conversion times inside the gateway in the test set-up. ......................................................... 41
5.8 The frequency distribution of the execution times of message processing tasks inside the WM ECU. ......................................................... 42
5.9 Simulation model with path from ECU 2 to the WM ECU marked by red line. ......................................................... 43
5.10 Arrival time and latency variation for different period of CAN message transmission. ......................................................... 44
5.11 Frequency distribution of the latencies of messages inside the WM ECU from the POOSL model. ......................................................... 44
5.12 Frequency distribution of the latencies of the messages covering the path from CAN to Ethernet. ......................................................... 45
5.13 Frequency distribution of latencies of messages inside the WM ECU simulation model for equal period of message arrival and output. ........ 45
5.14 CAN message transmission with and without the average jitter in the period of generation. ......................................................... 46
5.15 World model ECU timings with and without jitter in CAN transmission. ......................................................... 46
5.16 Frequency distributions of latencies of messages across the WM ECU for different number of messages. ......................................................... 47
5.17 Frequency distributions of end-to-end latencies of test 1 for simulation model (left) and test set-up (right). ......................................................... 49
5.18 Percentage contribution of the model components in the end-to-end latency distribution. ......................................................... 49
A.1 POOSL data expression, taken from [6]. ......................................................... 59
A.2 POOSL process statements, taken from [6]. ......................................................... 60
C.1 Frequency distribution of end-to-end latencies of messages in test 1 on POOSL model with jitter having zero mean. ......................................................... 63
C.2 Frequency distribution of end-to-end latencies of high priority (left) and low priority (right) messages in test 2 on POOSL model with jitter having zero mean. ......................................................... 63
C.3 Frequency distribution of end-to-end latencies of the messages in test 2 on simulation model (left) and test set-up (right). ......................................................... 64
C.4 Frequency distribution of end-to-end latencies of high priority (left) and low priority (right) messages in test 3 on POOSL model with jitter having zero mean. ......................................................... 64
C.5 Frequency distribution of end-to-end latencies of messages in test 3 on simulation model (left) and test set-up (right). ......................................................... 65
C.6 Frequency distribution of end-to-end latencies of high priority (left) and low priority (right) messages in test 4 on POOSL model with jitter having zero mean. ......................................................... 65
C.7 Frequency distribution of end-to-end latencies of messages in test 4 on simulation model (left) and test set-up (right). ......................................................... 66
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Comparing analytical and simulation-based methods</td>
<td>16</td>
</tr>
<tr>
<td>5.1</td>
<td>List of tests performed on the complete test set-up and the simulation model.</td>
<td>33</td>
</tr>
<tr>
<td>5.2</td>
<td>List of tests performed on CAN test set-up.</td>
<td>35</td>
</tr>
<tr>
<td>5.3</td>
<td>End-to-end latency results of the tests on the simulation model. All the</td>
<td>47</td>
</tr>
<tr>
<td></td>
<td>values are in milliseconds.</td>
<td></td>
</tr>
<tr>
<td>5.4</td>
<td>End-to-end latency results of the tests on the test set-up. All the values</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td>are in milliseconds.</td>
<td></td>
</tr>
<tr>
<td>5.5</td>
<td>Confidence intervals for simulation and test set-up results.</td>
<td>48</td>
</tr>
<tr>
<td>B.1</td>
<td>Results of tests performed on the CAN simulation model.</td>
<td>61</td>
</tr>
<tr>
<td>B.2</td>
<td>Results of tests performed on the CAN test set-up.</td>
<td>61</td>
</tr>
</tbody>
</table>
List of Acronyms

ABS  Anti-lock Braking System
ACC  Adaptive Cruise Control
AEB  Automatic Emergency Braking
ADAS Advanced Driver Assistant Systems
AUTOSAR AUTomotive Open System ARchitecture
CACC Cooperative Adaptive Cruise Control
CAN Controller Area Network
CDF  Cumulative Distribution Function
CRC  Cyclic Redundancy Check
CSMA/CD Carrier Sense Multiple Access with Collision Detection
CSMA/CR Carrier Sense Multiple Access with Collision Resolution
DES  Discrete Event Simulation
DLC  Data Length Code
ECU  Electronic Control Unit
EDF  Earliest Deadline First
FCW  Forward-Collision Warning
FIFO  First In First Out
HMI  Human Machine Interface
HU   Head Unit
IAT  Inter-Arrival Time
ILP  Integer Linear Program
IP   Internet Protocol
LDW  Lane-Departure Warning
LIN  Local Interconnect Network
LKS  Lane Keeping Systems
MAC  Media Access Control
MOST  Media Oriented Systems Transport
OEM  Original Equipment Manufacturer
OSEK  Open Systems and their Interfaces for the Electronics in Motor Vehicles
POOSL  Parallel Object-Oriented Specification Language

pmf  probability mass function
SFD  Start of Frame Delimiter
TCP  Transmission Control Protocol
UDP  User Datagram Protocol
I would like to thank my supervisor Dr Arjan van Genderen for his assistance during the thesis. I would like to thank Dr Mladen Skelin at the Technical University of Eindhoven and my supervisors at TNO, Dr Arturo Tejada Ruiz and Ir. Geert Verhaeg for their supervision. I would also like to express my gratitude to TNO Automotive, Helmond for giving me the opportunity to do my MSc thesis in TNO and assisting it both academically and financially.

ISHU GOEL
Delft, The Netherlands
November 20, 2018
1.1 Motivation

Vehicles are an inevitable part of human life. In the past few decades, there has been a significant increase in the number of vehicles sold across the globe. For example, the number of passenger cars registered in Europe rose from 163 million units in 1990 to 254 million units in 2015 [7]. Over this time, a significant number of functionalities have been added to vehicles by embedding an increasing number of on-board electronics (including microcontrollers). Despite all the advancement in vehicles over the years, road accidents are still a major cause of death of humans. In the year 2016 alone, around 25,500 people died in road accidents in Europe [8]. In spite of all the traffic regulations, the number of road fatalities have not decreased significantly.

The automotive industry is taking various steps to increase the safety of passengers. These steps include Advanced Driver Assistant Systems (ADAS) functionalities, self-driving cars and connected cars. ADAS was introduced to help drivers to drive safely and efficiently. Features provided by ADAS includes Forward-Collision Warning, Automatic Emergency Braking, Lane-Departure Warning, Adaptive Cruise Control (ACC), electronic stability control, and so on. Implementing these functionalities so that all the functional and temporal requirements set by the system designers are satisfied is challenging.

Most of the ADAS functionalities are safety critical and should not only be functionally correct but also temporally correct. Temporally correct implementation means that ADAS functionalities need to respond either at a specific time instant or within a required time interval. Violating the timing requirements could risk the safety of the driver. For example, consider a car (host) using ACC functionality to maintains a constant distance with respect to the vehicle in front of it. If the car in front applies its brakes, the host needs to respond in time, otherwise a collision might occur. Some non-safety critical applications, like the infotainment system, also need to meet the timing requirements to ensure their quality of service and enhance driver comfort. Every ADAS functionality has its challenges for timing analysis and these challenges increase when the functionality is spread over more than one vehicle like in Cooperative Adaptive Cruise Control (CACC) [1] due to a more complex hardware architecture.

ADAS algorithms rely on environment data, so the implementation of ADAS functionalities requires input from the sensors like radar, lidar, ultrasonic sensor, cameras, and night-vision devices and so on [9]. Apart from sensors, it also requires various ECUs to perform processing and communication buses to communicate between various
CHAPTER 1. INTRODUCTION

components of the system. Presence of so many components increases the complexity of the hardware and software architecture of the vehicle. Determining end-to-end latencies of different signal paths inside an ADAS system becomes challenging because of the complex interaction patterns between the hardware and software components. To fulfill the timing requirements, these end-to-end latencies should remain within the provided bound. The analysis in this thesis is not limited only to worst case latency but to obtaining a probability distribution of latency. This is because using only worst case latency analysis leads to over-dimensioning or over-designing of the system as the worst case latency only occurs under extraordinary circumstances. For a system like CACC, determining end-to-end latencies becomes more challenging because of the presence of heterogeneous communication buses and multiple ECU’s with different temporal behaviours. This thesis focuses on determining end-to-end latency in system like CACC.

1.2 Problem statement and thesis goals

The need for determining the end-to-end latency as mentioned in section 1.1 provides the problem statement for this thesis:

Identify and implement a methodology to determine the distribution of end-to-end latencies of different signal paths inside ADAS systems.

This thesis focuses on one of the significant challenges faced by the automotive industry, which is to verify that the implementation of a particular functionality satisfies the temporal requirements set by the system designer. The goal is to determine a probabilistic distribution of end-to-end latency of different signal paths inside an ADAS system. The ADAS functionality of interest in this thesis is CACC. CACC functionality is described in detail in section 2.2. To test the chosen methodology, a simple test set-up is built. It displays the main behaviours and timing characteristics that a TNO’s full-fledge CACC implementation has, but can be more easily modified and instrumented.

The main goals of this thesis are:

1. Analyse various methods used in the automotive industry to perform timing analysis in ADAS systems.
2. Identify the method most suitable to determine the distribution of end-to-end latencies of different signal paths inside an ADAS system like CACC.
3. Implement the selected method to obtain a frequency distribution of end-to-end latency of different paths inside the ADAS system.
4. Validate the implementation of the selected method using a hardware test set-up.
5. Reflection on how different parameters in an automotive system affect a certain temporal metric.
6. Provide some guidelines on how to design a system in such a way that it provides all the required timing information.
1.3 Thesis outline

The thesis is organised in the following way:

In Chapter 2, the basic terminologies and fundamentals related to ADAS systems and timing analysis are explained. This chapter provides an overview of the CACC system. It also lists some of the challenges faced in performing timing analysis in the ADAS systems.

Chapter 3 describes some already available analytical and simulation-based methods used to determine end-to-end latency distributions inside ADAS system. This chapter also presents a comparison of the two methods and mentions the reasons for selecting the simulation-based method.

In Chapter 4, the approach taken for implementing the selected method is mentioned. This chapter also describes the implementation steps of both the test set-up and the simulation model.

Chapter 5 outlines the experiments performed on both the test set-up and the simulation model. It provides the results of the tests performed on the individual components as well as the whole system for both the hardware and simulation models. It also compares the results of both the models and provides reflections on the results.

Finally, Chapter 6 provides a summary of the work done and the results achieved in this thesis. The main contributions of this thesis are listed along with some recommendations on how to design an ADAS system in such a way that it provides all the required timing information. Some suggestions on the future work that could be done based on the work done in this thesis are also made.
In this chapter, the basic terminology and fundamental concepts related to ADAS systems and timing analysis are explained. First, in Section 2.1, some important timing metrics used to verify the temporal correctness of a system are defined. Second, in Section 2.2, an overview of a general ADAS functionality along with a small description of the individual components is provided with the help of an example of the CACC system. Finally, Section 2.3 lists some of the challenges faced in performing timing analysis in ADAS systems.

2.1 Timing metrics

Some of the important timing metrics used to verify the temporal correctness of a system are described below.

- **Execution time**: The time spent by a system in executing a task using its processor resources is called execution time of that task.

- **Response time** is defined as the amount of time taken by a system to process a request after it has been received. For example, the time between the activation and the completion of a task. In addition to execution time, response time includes the time spent in blocking and interference by other tasks.

- **Latency** is the time difference between the stimuli and output of a request. The latency of a request includes the response time for processing the request and the time the request takes to travel over the network. Latency is similar to response time when measured for an individual Electronic Control Unit (ECU) and is often referred to as response time instead of latency. For this thesis, we define latency as the time difference between two correlated events, i.e. an event which initiates another event. For example, the time taken between pressing the brake pedal and the time at which brakes are applied to a vehicle.

- **Throughput** is defined as the number of units of data transmitted/computed over a channel in a given amount of time. For example, in an infotainment system, the amount of data transferred per second during a video stream specifies the throughput of the system.

- **Jitter** is defined as variations in phase position, period and duty cycle of a timing signal. For example, the difference between any one clock period and average clock period is known as clock jitter.
Latency analysis provides us with three metrics: best case, worst case and average case latency. Worst case latency analysis is generally performed to ensure that tasks meet hard real-time constraints. Best case latency analysis is done to obtain the minimum time in which a task could be done and average case latency analysis provides the average time taken by a task to complete.

2.2 ADAS

An example of an ADAS functionality taken as a test case in this thesis is Cooperative Adaptive Cruise Control (CACC). CACC is an extension of Adaptive Cruise Control (ACC). A vehicle equipped with ACC adjusts its relative speed with respect to the vehicle in front of it while maintaining a desired inter-vehicle distance. ACC works based on the information collected from on-board sensors like radar, lidar and camera. The reliance on the radar means that an ACC-enabled vehicle can only react to changes in the behaviour of the vehicle in front after they happen. As a consequence, the inter-vehicle distance cannot be made smaller than that covered by the following vehicle in about 2 seconds [1].

CACC solves this problem by allowing the front vehicle to transmit its intended actions to the following vehicles via wireless communication. It enables the following vehicle to react sooner to changes in the preceding vehicle’s speed and acceleration. It can help to reduce the inter-vehicle distance up to 0.3 seconds [1]. CACC helps to increase road capacity, minimise fuel consumption and emissions, and enhance driver comfort. Figure 2.1 shows CACC behaviour in a truck platooning application. A truck platoon is a road-train of two or more trucks that uses CACC functionality.

![CACC-equipped trucks](image)

**Figure 2.1:** CACC-equipped trucks.

In the CACC system, vehicles drive at very small inter-vehicle time gaps. In such a set-up, velocity disturbances after braking or accelerating can be amplified in the upstream direction. The notion of string stability covers the disturbance evolution across a string of vehicles. In string-stable behaviour, disturbances attenuate in the upstream direction [1]. For CACC to work correctly, the following vehicle should receive and respond to the information obtained from the preceding vehicle in a required time...
duration. The verification of this temporal property of the CACC system is challenging because of its complex hardware and software architecture.

An example of the hardware architecture of TNO’s implementation of CACC is shown in Figure 2.2. The architecture presented by light blue colour denotes the lead vehicle and architecture presented by green colour denotes the following vehicle. The end-to-end path followed by a message generated by the brake pedal of the lead vehicle to affect the actuator of the following vehicle is shown by red line.

Figure 2.2: CACC architecture and message flow in a CACC architecture.

The description of some of the components used in general ADAS system (in CACC as well) is provided below:

1. **ECU:** An ECU in automotive electronics is an embedded system that controls one or more of the electrical systems or subsystems in a vehicle. Some of the examples of the ECUs present in the vehicle are Engine Control Module, Transmission Control Module, Brake Control Module, and Body Control Module. A high-end vehicle these days contains around 80 to 100 ECUs with several million lines of software code [10].

2. **Sensor:** A modern vehicle is equipped with a wide range of sensors including brake sensors, wheel speed sensors, pressure sensors, temperature sensors and so on for controlling various functionalities. For example, radar is used to detect objects and determine their range, angle and velocity. A radar emits radio waves which are reflected from an object. The reflected waves received from the radar is called an echo and is used to determine the direction and distance of the object.

3. **Communication bus:** Different components of an ADAS system like sensors, ECUs, gateway and so on are connected using communication buses. These buses have requirements like the assurance of message delivery, fast transmission speed, ability to resolve message conflicts and other characteristics. Commonly used communication buses in vehicles are Controller Area Network (CAN), Local Interconnect Network, Media Oriented Systems Transport, FlexRay, etc. For past few years, Ethernet use in the automotive has also started. Among all the available communication buses, CAN and Ethernet buses are the ones used in CACC. A small description of both these buses is provided later in chapter 4.
4. **Gateway:** A gateway is a networking hardware which allows data to flow from one communication bus to another. The gateway used in CACC connects CAN and Ethernet buses. While converting a CAN frame to an Ethernet frame, the gateway wraps the CAN frame in the data bytes of the Ethernet frame and uses the configured source and destination address. For sending a message from Ethernet to CAN bus via a gateway, the required CAN frame must be wrapped in the data bytes of the Ethernet frame before sending it to the gateway. The gateway unwraps the CAN frame and transmits it over the CAN bus.

In CACC, as shown in Figure 2.2, inputs from various sensors, brake pedal and MK5 (Wi-Fi) are received over the CAN bus and passed on to Ethernet bus via a gateway. All the messages from input gateways go to the World Model ECU. The World Model ECU, referred as *WM ECU* in this report, uses these input values to estimate the position, velocity, acceleration and so on of the vehicle in the front of it. A detailed description of the working of the WM ECU is provided in chapter 4. The WM ECU transmits the calculated estimates to the control ECU. The control ECU, based on the world model estimates, decide to increase or decrease the velocity of the vehicle and sends the control signals to motors and actuators via the output gateway. The predetermined velocity and acceleration of the front vehicle are also sent to the following vehicle over wireless communication via output MK5. The signal received at the input MK5 of the following vehicle goes through the same path of CAN bus, gateway, Ethernet bus and ECUs before it reaches the actuator.

To ensure a reliable CACC functionality, the end-to-end latency of the critical paths of the CACC system needs to be determined. The end-to-end latency of the path from the break/acceleration pedal in the preceding vehicle to the actuator of its follower provides a bound for the inter-vehicle distance [1]. End-to-end latencies for computations that span over several ECUs and communication buses are a function of task response times, message response times, and communication delays. The end-to-end delay and reliability of the channel are the dominant factors to determine what a safe inter-vehicle distance is. A graph of the maximum end-to-end delay versus spacing delay for a string stable behaviour in a CACC system is depicted in Figure 2.3.

In Figure 2.3, the term on the y-axis represents maximum end-to-end latency for the critical path of brake signal in CACC system and the term on the x-axis represents the time headway or spacing delay, the time vehicle takes to stop after brakes are applied. In particular, the shorter the desired inter-vehicle distance is, the smaller the worst case latency must be. Using only worst case latency analysis leads to over-dimensioning or over-designing of the system because the worst case latency only occurs under extraordinary circumstances. Hence, it is important to determine distributions of latencies on the critical paths in the CACC system.
2.3 Challenges of timing analysis in ADAS systems

Some of the challenges in determining latency in the ADAS components and in the complete system are:

- **Latency inside ECUs:** Different ECUs have different processing times which may vary based on the algorithms they execute and the hardware resources they have. Some ECUs can use a shared resource which can lead to tasks being blocked or delayed. The response time for different tasks on an ECU depends on the scheduling policy used in the ECU. Scheduling policies can be preemptive or non-preemptive. In preemptive scheduling, a running task can be preempted by some other task while in non-preemptive scheduling, a running task cannot be preempted by any other task before its termination. Some examples of scheduling policies are Earliest Deadline First (EDF), First In First Out (FIFO), Deadline Monotonic (DM) and so on. Scheduling policies can take various factors into account while like task priority level, task activation time, task deadline, task duration, nature of activation of the task being periodic or event-driven.

- **Heterogeneous bus systems:** Different communication buses are used at different places in ADAS systems. Each communication bus has its message types, speed, bandwidth and scheduling policy or method of handling the messages. If more than one node connected to the bus tries to send message on the bus at the same time, then the scheduling policy of the bus decides which message will go first. Hence, the latency of the messages is dependent on the scheduling policy used in the bus. Some scheduling policies classify messages or nodes into different priorities and higher priority nodes or messages are given preference. For systems having more than one type of communication bus, to determine the latency of a message to pass through the buses becomes challenging.

- **Synchronisation of components and clock drift:** Synchronisation of components is required for determining the time taken by a message to travel between components. In case of a distributed system, different ECUs run with their quartz crystal...
clocks and can have a clock drift of 2400 ms over a period of 24 hours [11]. These clock drifts can result in the variation of execution times of tasks and can impact the end-to-end latency of the system.

- **Software complexity:** Software architecture is designed based on the hardware architecture and consists of an operating system, communication bus drivers and application software [12]. Automotive software ranges from entertainment to safety-critical real-time control software. Also, many functionalities are scattered across automotive subsystems which makes software design more challenging.

- **Integration of subsystems:** While car manufacturer or Original Equipment Manufacturer (OEM) provide the final product to the customer, the implementation of most of the subsystems is outsourced to Tier 1 and Tier 2 suppliers [13]. These subsystems are designed based on the requirements provided by OEMs which includes performance requirements, but timing and synchronisation properties with other subsystems are either not correctly defined or partially defined [13]. Integration and verification of such subsystems is challenging.

Most of these challenges are also present in determining the distribution of end-to-end latency distributions of critical paths in CACC system. Some of the methods used to overcome these challenges are discussed in the next chapter.
Existing methodologies for timing analysis of automotive systems

Chapter 2 discussed some challenges in determining the distribution of end-to-end latencies of various signal paths in ADAS systems. In this chapter, some of the available methods are examined to identify a suitable solution to the problem stated in the Chapter 1. These methods must be able to capture the complex interactions between software and hardware components as to provide a tight bound for end-to-end latency estimates (as to avoid over-dimensioning of the system). These methods should be able to provide latency information at the system level as well as component level.

Research has been done in determining the timing behaviour of a set of tasks running on a single ECU, individual communication buses, systems containing various ECUs connected via a single communication bus. Research is also available for systems containing more than one type of communication buses [14] [15]. The available analysis methods can be broadly classified into two classes: analytical and simulation-based.

Section 3.1 describes analytical approaches used to perform timing analysis in the systems containing a single type of communication bus. In Section 3.2, various simulation-based approaches used for timing analysis of the same kind of systems are described. Section 3.3 compares the analytical and simulation-based methods.

### 3.1 Analytical methods

These methods typically use a set of mathematical equations along with some specific assumptions, about the system under consideration, to derive formulas that describe the timing properties of the system based on the properties of its components. Paper [16] discuss the schedulability analysis of CAN messages, and papers [2] and [17] provide stochastic and statistical analysis to compute the probability distributions of CAN message response times. Paper [18] present a timing analysis technique to obtain bounds for complete sensor-to-actuator chain in an Ethernet bus based automotive architecture. Further in this section, papers [16] and [2] are discussed.

The work in [16] suggests a method for optimal priority assignment for Controller Area Network (CAN) messages. This paper challenges the recognised schedulability analysis of CAN messages proposed by Tindell [19] and provides a revised schedulability analysis. A system comprising of various nodes having a static set of real-time messages connected to CAN bus is assumed for analysis. The worst case response time, $R_m$, of a CAN message $m$ is given by the following equation:
where $J_m$ is queuing jitter (time between the message being initiated and message being put in queue of transmitter), $C_m$ is the transmission time of the message and $w_m$ is the queuing delay (time that message has to wait in the CAN controller before starting transmission on the bus) of the message. The queueing delay ($w_m$) consists of blocking ($B_m$) due to lower priority message being transmitted on the bus and interference due to higher priority messages. The maximum value of blocking time can be given by the maximum of the transmission time of the messages with priority lower than $m$. The following equation for determining worst case queueing delay ($w_m$) was given by Tindell [19].

$$w_m = B_m + \sum_{\forall k \in \text{hp}(m)} \left\lfloor \frac{w_m + J_k + \tau_{\text{bit}}}{T_k} \right\rfloor C_k.$$  \hspace{1cm} (3.2)

where $\text{hp}(m)$ is the set of higher priority messages, $J_k$ is the queuing jitter of higher priority messages, $\tau_{\text{bit}}$ is the transmission time for a single bit and $T_k$ is the period of the message or the inter-arrival time of the message. The above equation can be solved by using the recurrence relation given in the below equation:

$$w_{m}^{n+1} = B_m + \sum_{\forall k \in \text{hp}(m)} \left\lfloor \frac{w_m^{n} + J_k + \tau_{\text{bit}}}{T_k} \right\rfloor C_k.$$ \hspace{1cm} (3.3)

The above relation has a starting value of $w_m^0 = B_m$ and the recurrence relation iterates to $w_m^{n+1} = w_m^n$ as shown in Tindell [19]. The worst case response time of the first instance of the message in the busy period is given by $J_m + w_m^{n+1} + C_m$. This analysis is done based on the assumption $D_m(\text{deadline}) \leq T_m$ which means that if the message $m$ is schedulable, then the busy period of message $m$ will end before $T_m$. This assumption is correct only for fixed priority preemptive scheduling in which no higher priority message will be waiting for transmission before the transmission of message $m$ is complete. However, for fixed priority non-preemptive scheduling, a higher priority message could be waiting for transmission when the transmission of message $m$ is complete and thus the busy period can be larger than $T_m$.

Paper [19] claimed that deadline monotonic assignment is optimal for assigning priorities to CAN messages. However, tests performed in paper [16] revealed that this analysis gave correct worst-case response times only for highest and second highest priority messages. The paper [16] proved the priority assignment algorithm proposed by Audsley [20] to be the optimal algorithm for scheduling fixed priority non-preemptive CAN messages.

The work in [2] provide the theory for the probabilistic analysis of the end-to-end latency for periodically activated tasks and messages. The model assumed to contain
some ECUs connected via CAN network. Tasks were scheduled using priority-based scheduling as in Open Systems and their Interfaces for the Electronics in Motor Vehicles (OSEK) [21] for ECUs and messages are transmitted over the CAN bus. Task computation on each ECU is triggered via a periodic activation signal from a local clock. The dataflow model of an automotive architecture is shown in Figure 3.1.

![Dataflow model of an automotive system and its end-to-end latency](image)

**Figure 3.1:** Dataflow model of an automotive system and its end-to-end latency, taken from [2]

In Figure 3.1, \( \{o_1, o_2, ..., o_n\} \) is the set of objects representing the computation and communication functions of the system, \( \gamma_i \) represent the ECU resource on which a task \( (T_i) \) of period \( (T_i) \), execution time \( (\varepsilon_i) \), initial phase \( (O_i) \) and priority \( (P_i) \) is executed. \( (\varepsilon_i) \) is a discrete random variable of a probability distribution \( f_{\varepsilon_i} \) of granularity \( \tau \). For each resource \( \gamma_k \), its hyperperiod \( H_k \) is defined as the least common multiple (lcm) of the periods of all the objects executed on it. The edges \( E = \{e_1, e_2, ..., e_m\} \) signify the data transfer between objects and a path from \( o_i \) to \( o_j \) is denoted as \( \Pi_{i,j} \). In the top part of Figure 3.1, source object \( (o_1) \) represents detection of an external event and sink object \( (o_7) \) represents the actuation output while \( \gamma_1 \) and \( \gamma_3 \) represent ECUs and \( \gamma_2 \) represent CAN bus. As shown in the lower part of Figure 3.1, the end-to-end latency of a path from \( o_1 \) to \( o_7 \) consists of initial sampling delay, local delay, remote delay and response time \( (R_n) \) of the last task. The sampling delay, \( Z_{1,2} \), is the time difference between occurrence of the external event \( (o_1) \) and the activation of the task \( (o_2) \). The local delay includes the response time of a task and the communication delay from one task to another inside a ECU and remote delay in the time taken in transmitting a message over a communication bus. The delay \( D_{i-1,i} \), represents the time difference from the activation of \( o_{i-1} \) and the activation of \( o_i \). The end-to-end latency, \( L_{1,n} \), for the path, \( \Pi_{1,n} \), is calculated using the below equation:
\[ L_{1,n} = Z_{1,2} + \sum_{i=2}^{n} D_{i-1,i} + R_n. \] (3.4)

With the assumption that \( Z_{1,2}, D_{i-1,i} \) and \( R_n \) are independent of each other, the probability mass function (pmf) of end-to-end latency \( f_{L_{1,n}} \) is the convolution of the pmfs of these terms as mentioned in below equation:

\[ f_{L_{1,n}} = f_{Z_{1,2}} \otimes \sum_{i=2}^{n} D_{i-1,i} + R_n. \] (3.5)

There are some disadvantages of using this method for determining end-to-end latency. One is the long run-time because of the enumerative approach, and the other is the inability of the analysis to capture interdependence between various distribution parameters like sampling delay, local delay, remote delay and response time as they assumed them to be independent.

### 3.2 Simulation-based methods


The work in [3] provides simulation models for analysing the performance of a switched Ethernet-based, in-car network with different traffic classes and a general topology. The simulation is performed on the OMNET++ tool with the INET framework. As most of the ECUs are located either at the front or the rear of a vehicle, a double-star topology is designed for simulation of an ECU network in the car. The assumptions made for simulation set-up include switch processing time of 3 microseconds, transmission queue size as 1000 packets, IP-address configuration is considered to be static, and bandwidth of Ethernet is assumed to be 100 Mbit/s. In the experiment, two timing metrics, end-to-end delay and Inter-Arrival Time (IAT) are measured for the control data and the camera data sent to the Head Unit (HU). The HU is the centre piece of the car’s sound and information system and provides the user control over the vehicle’s information and entertainment media. The end-to-end delay and IAT provide information about the packet delay and packet loss observed due to overloading in the network. The service constraints for the maximum value of the end-to-end delay for the control data transmission and the driver assistance camera data transmission are 10 ms and 45 ms respectively. The results of the simulation model measuring Cumulative Distribution
3.2. SIMULATION-BASED METHODS

Function (CDF) of the end-to-end delay of data transmission from the control system and the camera are shown in Figure 3.2 and results of IAT for the same systems are shown in Figure 3.3.

![Figure 3.2: CDF of end-to-end delay of transmission of control data and camera data, taken from [3]](image1)

![Figure 3.3: CDF of IAT of transmission of control data and camera data, taken from [3]](image2)

It was observed that as the number of intermediate switches increases along the transmission path, the end-to-end delay increases. For both the control and camera data, the service constraints were violated. In the case of inter-arrival time, no packet loss was observed for the control data packets, but for the camera data packets, half of them had a packet loss. Hence, the paper concluded that to fulfil the requirements
of an Ethernet-based in-car network having a double-star topology, services must be prioritised.

3.3 Comparing simulation-based methods with analytical methods

In this section, a small comparison of analytical and simulation-based methods is done based on the methods discussed in Section 3.1 and 3.2 for an ADAS system like CACC. Both the methods have some advantages and disadvantages. A brief comparison of analytical and simulation-based methods is shown in Table 3.1. The points listed in the Table 3.1 could vary for other methods. Analytical methods guarantee performance limits. Analytical methods can cover all the corner cases while simulation-based methods cannot. The validity of simulation-based methods is limited to the input profiles used for the analysis [11].

<table>
<thead>
<tr>
<th>Properties</th>
<th>Analytical</th>
<th>Simulation-based</th>
</tr>
</thead>
<tbody>
<tr>
<td>Covers corner cases</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Scalability</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Model complex interactions</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 3.1: Comparing analytical and simulation-based methods

The downside of analytical methods is that they are rarely able to model complex interactions and state-dependent behaviours [23]. This results in too-pessimistic end-to-end bounds which lead to an over-dimensioned system. On the other hand, simulation-based methods have customizable modelling scope which allows them to model complex interactions in a system. Analytical methods cannot be easily scaled up according to the requirements of the system, though this is not true for all analytical methods. One more downside with some analytical methods is that they make assumptions which may not hold in reality, e.g. considering two random variables independent of each other or assuming no jitter in the system components. Some analytical methods are also limited to some defined distributions like Gaussian and uniform and do not provide good results if the values cannot be fit into a specific distribution.

Because of the increasing complexity of the automotive systems, simulation-based methods have gained more importance, especially in the industry where time to market is of the essence. CACC is a complex system with a fair amount of uncertainty involving complex component interactions in terms of task release jitter, variable task execution times, various scheduling algorithms and communication semantics. Hence, a simulation-based method is taken in this thesis for determining the end-to-end latency of different paths in the CACC system. The approach followed to complete the goal of this thesis is explained in the next chapter.
As mentioned in Section 3.3, simulation-based method was selected to determine end-to-end latency distribution in ADAS systems. In this chapter, Section 4.1 describes the approach taken for implementing the simulation method. Next, the experimental set-up is described in Section 4.2. Finally, the simulation model built in this thesis is explained in Section 4.3.

### 4.1 Approach

The simulation-based approach taken in this thesis is presented in brief in Figure 4.1. First, for a selected system, a hardware test set-up is made. The test set-up mimics the selected system and uses equipment used in the CACC system. The test set-up provides latency of the complete system as well as of the individual components. Then, simulation models of the individual components of the system are made. These models are then combined to make the simulation model of the complete system. This method will support modularity, re-usability and easy instantiation of the model. The models are combined based on the assumption that if the timing behaviour of the individual components of a system is known, then the timing behaviour of the complete system can be described by the simulation model of the complete system.

![Figure 4.1: A brief description of the approach taken in this thesis.](image)

The latencies of the individual components without any interference (within the component or from any other component) were measured from the test set-up. The
simulation model makes use of these measurements and accounts for the interactions within and between the components. Hence, provides the end-to-end latency of the system. Tests are performed on both the test set-up and simulation model. The results of the end-to-end latency of the simulation model are compared to the results of the test set-up in order to validate the simulation model.

The current implementation of the CACC system at TNO does not directly provide timing information in some parts of the system. For example, the time taken by a message to pass through the WM ECU cannot be measured. While building a simulation model for a system, it is required to test the results of the simulation model under various configurations of sensors, like variable data length of messages and periodic or event-driven activation. There is limited access to change the configuration of the sensors (and other parameters) in the already constructed systems which restricts the validation of simulation model.

Because of these limitations, a new system is designed which covers the timing aspects of the CACC system and provides complete access to change the timing characteristics as needed. The main three timing aspects of the CACC system are the transmission of a message from sensors to the gateway via CAN, transmission of a message from the gateway to world model ECU via Ethernet and transmission of a message through world model and control ECU. Hence, a small part of the CACC system is selected which covers these three timing aspects. The selected system is marked by the red colour box in Figure 4.2.

The description of some of the main components which affect the end-to-end latency in the CACC system is provided below:

1. **CAN bus**: Controller Area Network (CAN) is a serial communication protocol
which efficiently supports distributed real-time control [24]. It is the most widely used communication protocol in the automotive industry. Its main properties are:

- It is an asynchronous multi-master serial bus with one logic bus line and equal priority nodes.
- The number of nodes can be changed dynamically without disturbing other nodes in the network.

- **Frame formats:** Four types of frames are available in the CAN protocol namely data frame, remote frame, error frame and overload frame. The layout of a standard data frame is shown in figure 4.3. The message identifier is 11-bit long for a standard frame and 29-bit long for an extended frame. The control field contains Data Length Code which specify the number of data bytes present in the message. The data field can have 0 to 8 data bytes in a frame. The Cyclic Redundancy Check (CRC) field is used to detect transmission errors. The acknowledge bit is set by any node which receives an error-free frame. The maximum size of a standard CAN data frame can be 108 bits.

- **Schedulability:** The bus access in the CAN is handled via Carrier Sense Multiple Access with Collision Resolution (CSMA/CR) protocol with arbitration on message priority. The identifier in every message determines the priority of the message (smaller identifier has a higher priority). Every message on the bus is broadcast to all the connected nodes. Each node stores or discards the received message based on the value of the identifier. A node which wants to transmit a message, first, checks for the bus to be in idle state (i.e. no message on the bus), and then transmits the message. If more than one node attempt to transmit the message on the bus at the same time, the

![Figure 4.3: Standard CAN data frame, taken from [4]](image-url)
collision of the messages is avoided by bitwise arbitration. The node with the smallest value of the message identifier wins the arbitration and completes the transmission of the message. Other nodes wait for the bus to be idle again and retry to transmit the message.

2. **Ethernet:** The need for Ethernet in automotive systems has come as the bandwidth of common bus field technologies like CAN, Local Interconnect Network, and FlexRay has been exhausted by infotainment and ADAS functions. The Ethernet bus technology fulfils this demand for high bandwidth and provides low-cost components [18].

- **Frame format:** The layout of a standard IEEE 802.3 Ethernet frame is shown in Figure 4.4

![IEEE 802.3 Ethernet Frame](image)

Preamble and Start of Frame Delimiter have defined values and are used for synchronisation between sender and receiver and indicates the start of the frame. The destination and the source addresses are both 6 bytes long and contains the Media Access Control (MAC) address of the destination and source machine respectively. The length field specifies the number of data bytes present in the Ethernet frame. The data field contains the data bytes of the frame and can have between 46 and 1500 data bytes in a frame. If the data length is less than 46 bytes, then zero padding is used to bring the length up to 46 bytes. The FCS field contains 32-bit CRC code to check for transmission errors.

- **Schedulability:** Ethernet uses Carrier Sense Multiple Access with Collision Detection (CSMA/CD) access control to determine which node can transmit data on the bus. Every node on the network has equal priority, and bus access is provided on a first-come, first-serve basis. Any node that wishes to transmit a message first waits for the bus to be in the idle state. If two nodes start transmitting at approximately the same time and a collision is detected, then both the nodes stop transmitting and wait for a semi-random amount of time before transmitting the message again. Nodes keep trying to transmit the message until they are successful and wait more amount of time with every consecutive collision.

3. **World Model ECU:** The WM ECU receives the input from gateways in an event-driven fashion and transmits the world model estimates to control ECU.
periodically. The functioning of the WM ECU is shown in figure 4.5.

![Figure 4.5: The functioning of the WM ECU.](image)

The WM ECU input sampler accepts inputs as they come from various sensors and perform sensor fusion if required and sends them to the target tracker. The target tracker uses these values to recalculate world model estimates (the position, velocity and acceleration of the vehicle in front) and transmits them to the output sampler as soon as they are ready. The output sampler stores the world model estimates received from target tracker in a buffer along with their arrival time as timestamps. The output sampler generates a message periodically by taking a value of world model estimates from the buffer. The period of the output sampler is 10 ms. At any given time, the estimates from the buffer are chosen based on the timestamps. The output sampler picks the estimates having the timestamp smaller and closest to the time of message generation. On the selected value, an estimation is performed to predict the output values of the target tracker for the current time. These world model estimates are then sent via an output message from the WM ECU. An example of the message flow inside the WM ECU is presented in Figure 4.6. The green arrows at the input sampler denote the arrival time of the message at the WM ECU, the yellow arrows at the target tracker denote the time at which estimates of the target tracker are put in the buffer of the output sampler and the red arrows at the output sampler indicates the time at which messages are sent from the WM ECU.

As shown in Figure 4.6, the arrival time of messages at the input sampler could be random, but the output of the output sampler is periodic. The time difference between the green arrow of input sampler and respective yellow arrow at target tracker denotes the execution time of that message. The latency of a message through world model ECU is the time between its arrival at the input sampler and time at the output of output sampler. For example, the latency for the message received at time $c$ will be $(20 - c)$ ms. The latency of all the messages received at input sampler should be calculated even if more than one messages are received during one period of output sampler. For example, for messages received at time $a$ and $b$, the latency will be $(10 - a)$ and $(10 - b)$ ms respectively. A message could have latency more than the period of output sampler. For example,
message arriving at input sampler at time \( d \), the execution time finishes after an output at time \( 40 \) has been already generated. This message is then taken into account for next output message, and its latency will be \((50 - d)\) ms.

The new system used for determining end-to-end latency is shown in Figure 4.7. The ECU 1 and ECU 2 can generate CAN messages of different priority and data length at different time intervals which could be periodic or aperiodic. The gateway generates an Ethernet message for each received CAN message and sends it to the WM ECU. The WM ECU behaves similar to the WM ECU in the Cooperative Adaptive Cruise Control (CACC) system regarding timing as described above, i.e. it reads event-driven inputs and generates periodic output.

Figure 4.6: Message flow inside World model ECU.

Figure 4.7: Selected system. WM ECU denotes World Model ECU.
The test set-up and simulation model made for this new system are described in the following sections.

4.2 Test set-up

The test set-up is built using some of the equipment used in an actual CACC system at TNO. These equipment include Axiomtek tBOX312-870-FL [25], PCAN-USB [26] and PCAN-Ethernet Gateway DR [27]. Axiomtek tBOX312-870-FL, referred to as Axiomtek in this report, is an embedded system that runs an embedded Operating System like Windows 7 embedded or Linux, and provides many I/O interfaces and LAN and USB ports. The PCAN-USB is a CANUSB by the PEAK-System company. A CANUSB adaptor can be plugged into any PC USB Port and provides an instant CAN connectivity. The PCAN-Ethernet Gateway DR, referred to as gateway in this report, is a gateway provided by the PEAK-System company to connect CAN and Ethernet buses. Figure 4.8 depicts the diagram of the test set-up.

![Diagram of the test set-up](image)

Figure 4.8: Test set-up.

The CANUSBs are used to send and receive messages on the CAN bus. The CANUSBs operates via virtual ECUs running inside the Axiomtek. The gateway is connected between CAN and Ethernet bus. The WM ECU operates inside Axiomtek and receives input from the gateway via Ethernet. Subsequent subsections provide the details of the designing and setting up of the components of the test set-up.

4.2.1 CAN setup

ECU 1 and ECU 2 can generate CAN messages of data lengths between 0 and 8 bytes with a variable identifier at different baud rates. The message generation could be aperiodic or periodic with different periods. For writing a CAN message on the bus, the virtual ECU inside the Axiomtek first initialise the connected CANUSB with its device
id and the desired baud rate. After the device initialisation is successful, the desired CAN message is framed and written on the bus. After the writing of the message is complete, a CAN status of `CAN_ERROR_OK` is received from the CANUSB. Similarly, an ECU can start reading the CAN message after initialising the CANUSB device. The CAN status of `CAN_ERROR_QRCVEMPTY` is received while the ECU is waiting for the message and while it is reading the message. The CAN status of `CAN_ERROR_OK` is received when the reading of a CAN message is complete.

For timing analysis purpose, all the CAN messages were logged inside the Axiomtek. Before writing the CAN message on the bus, the ECU logs the message along with the current timestamp. Similarly, as soon as the reading of the message is complete, the current timestamp information is logged along with the received CAN message. These logs can be used to calculate the latency of a message going from ECU 1 to ECU 2 via CAN bus and provide the start time of the CAN message going to the gateway. A sequence number was also given to the CAN messages for easy calculation of their latency through gateway and Ethernet bus.

### 4.2.2 Gateway and Ethernet set-up

The gateway is connected to the Axiomtek using an Ethernet cable. For connecting more than one gateway or Axiomtek to Ethernet bus, a hub could be used. The gateway device used in this thesis is configured using a web interface. The gateway device has its pre-configured Internet Protocol (IP) address and MAC address which can be changed through a web interface.

For the device configuration, first, a route is set which could be either from CAN to IP or from IP to CAN. Second, the destination address of the device connected to the gateway through Ethernet is fed in the web interface along with the port number. Third, the communication protocol is chosen as either Transmission Control Protocol (TCP) or User Datagram Protocol (UDP). Finally, a CAN channel is selected, and the baud rate of the CAN channel is set. While the gateway device is running, it receives CAN messages from the CAN channel and wraps them inside an Ethernet message using the already configured IP addresses. The device receiving the CAN message from the gateway needs to unwrap the CAN message from the Ethernet message, and similarly, to send an Ethernet message to the gateway, it needs to create the Ethernet message in such a way that its data field contains a CAN message.

### 4.2.3 World Model ECU implementation

The WM ECU in the test set-up is similar to the WM ECU (shown in figure 4.5) in the CACC system regarding timing behaviour but not regarding processing algorithms. Figure 4.9 depicts the flowchart for the working of the WM ECU used in the test set-up.

The WM ECU receives the Ethernet messages coming from the gateway and extracts the CAN message wrapped inside them. From these CAN messages, it obtains the required information like message identifier, data length and data bytes. The input
4.2. TEST SET-UP

The WM ECU uses the received data to modify only one output variable. After updating the output variable, a small delay was provided which adds to the execution time of the ECU. This delay was added because, without this delay, the execution time of WM ECU will be very less compared to the execution time of the WM ECU in the CACC system. The timestamp at the end of delay for every message is also stored in the buffer with other message parameters. This timestamp helps in determining the execution time of a message inside the WM ECU.

To make the output of the WM ECU periodic, a thread was run in parallel to the algorithm receiving the messages. This thread creates a timer which calls a function periodically. It allows the output sampler to work in parallel to the input sampler. The WM ECU, first, waits for the period to finish and then prepares the output message using the time of the creation of message along with the recent value of the output variable. The output message also contains all the values of the timestamp, message id and execution time present in the buffer. The buffer is cleared after every output message generation and can be filled again by input sampler. The output messages are logged to provide information for the calculation of the execution times and latencies of each message inside the WM ECU. The timestamps put during the input and output
of the messages inside the WM ECU helps in determining the latency of the messages through the WM ECU.

### 4.3 Simulation model

The layout of the simulation model made for the new system (described in the Section 4.1) is shown in Figure 4.10. The components of the simulation model characterise the level of abstraction in the model where every component works as a black box characterised by an input-output latency. This input-output delay of the components is configured using the delay measured across the components in the test set-up. Simulation model accounts for the complex interactions between the components and provides the end-to-end latency of the system.

![Simulation model](image)

**Figure 4.10:** Simulation model.

Section 4.3.1 describes the features of the simulation tool used to design this model, and the subsequent sections describe the models of the components.

#### 4.3.1 Simulation tool

An Eclipse-based tool called PooslIDE [28] was used for implementing the simulation model. It uses a Discrete Event Simulation (DES), called Rotalumis [29], to simulate models made using Parallel Object-Oriented Specification Language (POOSL) language. In DES, a system operates as a discrete sequence of events in time. Simulation time proceeds with the occurrence of events and is not dependent on the hardware resources that it uses. POOSL [30] is a very expressive language to model concurrent hardware/software systems. It has a small set of powerful primitives and an unambiguous formal semantics in terms of mathematical axioms and rules. The formal semantics of POOSL forms the basis of Markov-chain based performance analysis techniques and run-time model
checking techniques for Formal Verification of correctness properties. An overview of architectural structure and syntax of POOSL language is provided in appendix A.

The PooslIDE tool has a unique feature that on the top of the POOSL model lies an observer that terminates the simulation once a desired confidence level for the timing values has been reached. The confidence level statistics provides a confidence interval and an estimate of sample size required for obtaining the confidence level. A confidence interval [31] is a type of interval estimate, computed from the statistics of the observed data, that might contain the true value of an unknown statistical parameter (e.g. mean). The confidence level represents the proportion of possible confidence intervals that contain the true value of the unknown statistical parameter. Doing performance analysis by simulation assumes that the execution of the model will eventually result in some rather steady state where the measured performance metric values remain at a fixed level [32]. POOSL provides library data classes for performance analysis which can be easily added to the POOSL model.

### 4.3.2 CAN POOSL model

The working of the CAN bus was explained in Section 4.1. The POOSL model is designed to behave the same way as a CAN bus would do. Figure 4.11 shows the composite structure diagram of the CAN POOSL model.

![Composite structure diagram of the CAN POOSL model.](image)

The number of ECUs connected to CAN bus can be changed. The working of the ECUs connected to the CAN bus model can be described by the processMessage method shown in Figure 4.12. The processMessage method, first, creates a CAN message by calling createMessage method. Next, it starts three concurrent tail-recursive methods sendMessage, receiveMessage and CheckAccuracyStatus. The parallel execution of these methods is done using the parallel composition statement, \{par S1 and ... and Sn rap\}, shown in lines 7 to 13 in the Figure 4.12. The parallel execution of sendMessage and receiveMessage methods ensures that the transmission and reception of the messages is uninterrupted.

While creating a message, along with message identifier and data, time of creation
CHAPTER 4. FRAMEWORK FOR TIMING ANALYSIS OF HETEROGENEOUS IN-VEHICLE NETWORKS

Figure 4.12: Working of the ECU models connected to the CAN bus model.

of the message (called as entryTime in the model) is also stored in the message. The CheckAccuracyStatus method checks if the latency of the messages fulfils the confidence level requirements. The time at which an ECU receives a message is stored as endTime in the model and is logged with the received message. The difference of the endTime and the entryTime provides the latency of the message. Figure 4.13 depicts the POOSL code for performing the arbitration on the CAN bus.

Figure 4.13: CAN bus arbitration code.

The CAN bus is modelled as a process class which receives messages from the connected ECUs and performs arbitration on the messages if required. A distribution of the latencies of the messages to go from one ECU to another via CAN bus without facing any delay or arbitration from the messages of any other ECU was derived from the test set-up. The simulation model accounts for the delay caused by the arbitration and by the time spent in waiting for the bus to be in the idle state. The simulation time proceeds in POOSL with the delay statement. A buffer named messageBuffer stores the messages received from different ECUs. First, the message with the minimum identifier value (highest priority) is selected from the buffer for transmission over the bus. The simulation time proceeds equivalent to the amount of latency (referred by
variable `transTime`) of the selected message and then the message is sent to all the connected nodes.

In case, a new higher priority message is sent to the CAN bus before simulation time of the selected message could start proceeding, then the new higher priority message will be transmitted first. To model such a scenario, the `delay` statement proceeding the simulation time of transmission of the selected message is executed inside an `abort` statement. The `abort` statement (on line number 19 in the Figure 4.13), `{abort S1 with S2}`, terminates the execution of S1 statement in case S2 starts executing. The statement, `Arb?message(n)`, on line number 21 in the Figure 4.13 denotes the reception of the new message. Hence, the `delay` statement of the selected message (on line number 20 in the Figure 4.13) gets aborted in case a new message arrives during the execution of the `delay` statement.

The variable `startTime` stores the simulation time before the `delay` statement starts execution and the `timeInstant` variable stores the simulation time at which the `delay` statement either executes completely or gets aborted by an incoming message. If a new message arrives before the `delay` statement could start its execution, i.e. `startTime = timeInstant` (if condition on line number 23 in the Figure 4.13), then the newly received message is stored in the message buffer along with the message that was selected for transmission. A new selection is made to identify the higher priority message and the transmission process is repeated.

If the `delay` statement executes completely without any disturbance, i.e. `timeInstant >= startTime + transTime` (if condition on line number 29 in the Figure 4.13), then the message is sent to the connected ECUs. If the `delay` statement (on line number 20 in the Figure 4.13) was aborted after some simulation time had already passed, it would mean that the new message arrived when the selected message already started transmission. In that case, the selected message completes the progress of its remaining simulation time (using `delay` statement on line number 32 in the Figure 4.13) and the new message is stored in the buffer to be considered for transmission later.

### 4.3.3 Ethernet POOSL model

The working of the Ethernet bus was explained in Section 4.1. The design of this model is similar to CAN POOSL model except for the arbitration process. The composite structure diagram of Ethernet POOSL model is also identical to the one depicted in Figure 4.11 replacing CAN bus with the Ethernet bus. The ECUs connected to Ethernet bus send and receive Ethernet messages in the similar way as they do in the CAN POOSL model. The test set-up provides a distribution of the transmission times or latencies of messages to go from the gateway to the Axiomtek via Ethernet bus without facing any interference. Figure 4.14 depicts the code for the arbitration inside the Ethernet bus model.

As the Ethernet bus transmits the messages in a First In First Out (FIFO) manner,
the simulation model starts the progress of the simulation time for the first message that it receives. The delay statement is executed inside the abort statement (line number 40-42 in the Figure 4.14) as in CAN model. In case, a new message arrives to the Ethernet bus before simulation time of the selected message could start proceeding, it would mean that a collision is detected. In that case, the delay statement will get aborted before it starts execution, i.e. startTime = timeInstant (if condition on line number 44 in the Figure 4.14). The detection of the collision is informed to the sendMessage method with true argument. The sendMessage method then sends the acknowledgement to both the transmitting ECUs that a collision is detected and both the ECUs wait for a semi-random amount of time before transmitting the message again.

If the delay statement gets aborted after some simulation time has already passed, it would mean that the new message arrived when the selected message already started transmission. In that case, the selected message completes the progress of its remaining simulation time (using delay statement on line number 48 in the Figure 4.13) and the ECU which tried to transmit the message waits for the delay statement to complete before transmitting the message again. The false argument inside the sendMessage method denotes that no collision was detected and the message could be transmitted to the connected ECUs.

4.3.4 Gateway POOSL model

The functioning of the gateway was explained in the Section 2.2. Gateway POOSL model performs the conversion of a CAN message into an Ethernet message. Figure 4.15 depicts the composite structure diagram of the POOSL model involving CAN and Ethernet bus connected through the gateway.

In Figure 4.15, the gateway POOSL model is depicted by the Gateway node in the centre. The CAN_bus and Ethernet_bus node denotes the POOSL model of the CAN and the Ethernet bus. The CAN_Gateway node connects the CAN bus to the gateway. It does not generate any message of its own and sends all the messages it receives to the Gateway node. The Gateway node wraps the received CAN message inside an Ethernet messages and send them to the Ethernet_Gateway node. The Gateway node, while preparing
4.3. SIMULATION MODEL

Figure 4.15: POOSL model showing CAN and Ethernet bus connected through gateway.

the Ethernet message, waits for the conversion time before sending the message to the EthernetGateway node. The EthernetGateway node connects the Ethernet bus to the gateway. It receives the messages from the gateway and sends them on the Ethernet bus and does not generate any message of its own. The conversion time of the gateway is obtained from the test set-up. The Gateway block stores the received messages in a buffer and perform conversion to Ethernet message in a FIFO manner. If a new message is received during conversion of a message, it is stored in the buffer and is converted later.

4.3.5 World model ECU POOSL model

The WM ECU POOSL model is designed to have the same timing behaviour as the WM ECU designed in the test set-up in the Section 4.2.3. The distribution of the execution time of the WM ECU is obtained from the test set-up and used in the POOSL model. The working of the WM ECU can be described by the processMessage method shown in Figure 4.16.

Figure 4.16: World model POOSL code.

The processMessage method starts three concurrent tail-recursive methods updatePeriod, receiveMessage and CheckAccuracyStatus. The updatePeriod method provides the value of simulation time of the output message to keep the output of the WM ECU periodic. The CheckAccuracyStatus method checks if the latency of the messages fulfils the confidence level requirements. The receiveMessage method
receives the messages from the gateway, extracts the required data and prepares the output message.

The `receiveMessage` method after receiving a message, first, logs the entry time of the message and checks if the message arrived from the `Ethernet_Gateway` node. If the message arrived from the gateway, it extracts the CAN message from the Ethernet message. From this CAN message, information required for latency calculation like message identifier, transmission time of CAN message, etc. are also extracted. The progress of the simulation time equivalent to the execution time of the message is done via the `delay` statement. The output message is then prepared using the output time provided by the `updatePeriod` method. The output message contains information of all the messages that arrived within a period. The output messages are logged with all the required timing information and latency for all the messages entering the WM ECU can be calculated using these logs.

The next chapter discusses the experiments performed on both the test set-up and simulation model explained in this chapter.
Experiments and Results

In this chapter, the experiments performed on the test set-up and the simulation model are described along with the results of these experiments. Simulation results are validated by comparing them to the results of the test set-up. These experiments are performed following the approach mentioned in Section 4.1. Section 5.1 outlines the test plan for performing the experiments and assumptions made before performing these experiments. Section 5.2 provides details on the results of the tests performed on the individual components of the test set-up. It also explains how the simulation model uses these results. Section 5.3 describes the results of the tests performed on the complete simulation model and test set-up for determining end-to-end latency. Finally, Section 5.4 provides some reflection on the results of the experiments performed in this thesis.

5.1 Test Plan

As mentioned in Section 4.1, experiments were performed in two steps. First, to obtain an input-output latency of each component in the model and then for the end-to-end latency of the complete model. The first step of the tests on the test set-up (described in the Section 4.2) includes determining the latency of a message sent from one node to another via CAN bus, determining the time taken by an Ethernet message to reach Axiomtek from gateway, time taken by gateway to convert a CAN message to an Ethernet message and the execution time of a message processing task inside the WM ECU. The second step of the tests on both the simulation model and the test set-up includes determining the end-to-end latency of the model under different scenarios and comparing the results of both the models. The tests performed in the first step along with their results are discussed in Section 5.2. The tests performed on the whole simulation model and the test set-up are outlined in Table 5.1.

<table>
<thead>
<tr>
<th>Test No.</th>
<th>No. of CAN nodes</th>
<th>Period (ms)</th>
<th>High Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>10 ms</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3 ms, 6 ms</td>
<td>3 ms</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>20 ms, 50 ms</td>
<td>20 ms</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>3-6 ms, 3-10 ms</td>
<td>3-6 ms</td>
</tr>
</tbody>
</table>

Table 5.1: List of tests performed on the complete test set-up and the simulation model.

All the parameters mentioned in the Table 5.1 are for the CAN nodes as they generate the signals from the sensors or the user input. The column No. of CAN nodes denotes the number of CAN nodes transmitting the messages on the bus. The transmission of CAN messages is periodic for all the tests. The column Period provides
the time interval at which the messages are transmitted from the CAN nodes. In case more than one CAN nodes are transmitting messages, the node sending the higher priority message is denoted by its period of transmission, i.e. for test number 2, the node transmitting the message at 3 ms period has higher priority. All the CAN messages used in these tests have 8 bytes of data and a baud rate of 125 kbps. The Ethernet messages have a baud rate of 100 Mbps and the WM ECU has an output period of 10 ms.

The test number 1 is a simple case in which only one CAN node transmits the messages. This test validates the working of the simulation model by comparing its end-to-end latency with the test set-up. The other tests have two transmitting CAN nodes and aims to observe the effect of interactions between the messages on the end-to-end latency. The transmission of CAN messages is kept periodic in most cases as the sensors and ECUs in the CACC system transmit messages periodically. The time intervals in test 3 are two minimum periods of CAN message transmission in the CACC system. The time intervals in the second test are chosen in order to have a higher degree of interference between CAN messages (non-preemptive priority-based scheduling). In test 4, CAN nodes does not have a fixed period of message transmission. The first CAN node can randomly select an integer value between 3 and 6 ms as a time interval between two consecutive CAN message transmission. Similarly, second CAN node can select an integer value between 3 and 10 ms as the time interval between message transmission. The last test is done to evaluate the validity of the model for message transmissions that are characterized by minimum and maximum inter-arrival times.

Some of the assumptions made on the test set-up for performing tests and for implementing the simulation model are mentioned below:

1. The bus utilisation for both CAN and Ethernet bus is less than 1 and no messages are lost on the bus.
2. The periodic generation of CAN messages has some jitter which does not vary with multiple experiments.
3. The data processing in the gateway is event-driven, where an event is considered to be raised when data arrives at the input interface of the gateway.
4. The conversion time of the gateway does not vary with the size of the CAN message.
5. Internal buffer of the WM ECU has enough capacity to not lose any incoming message.
6. The periodic generation of the output messages by the WM ECU has a negligible jitter.
7. The execution time of the processing done inside the WM ECU does not vary with the load on the Axiomtek.
8. The transmission time over the communication bus, the conversion time of the gateway and the execution time of the processing done inside the WM ECU are uniformly distributed within a certain interval.
5.2 Measurement of the latency of the components

In this Section, experiments performed for the measurement of the frequency distribution of latencies across different components of the test set-up are described. The method used to populate simulation model component using frequency distributions of the test set-up measurement is also discussed. The frequency distributions obtained from the simulation model are also compared with the test set-up. To compare the distributions, t-test [33] is used. The t-test is a type of inferential statistics used to determine whether there is a significant difference between the means of two groups of values. For comparing two groups of values by t-test, a value of hypothesized mean difference between the groups is selected along with an acceptable level of probability (that the mean difference occurred by chance). The t-test then confirms whether the value of the hypothesized mean difference between the provided groups holds or not.

5.2.1 Tests on CAN model

For measuring timing across the CAN bus in the test set-up, several experiments were conducted. For conducting these experiments, the gateway and the Ethernet bus were removed from the test set-up shown in the Figure 4.8. The tests performed on the CAN test set-up are listed in Table 5.2.

<table>
<thead>
<tr>
<th>Test No.</th>
<th>No. of nodes</th>
<th>Baud rate (kbps)</th>
<th>Data length (in bytes)</th>
<th>Period (ms)</th>
<th>High Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>100, 125, 250, 500</td>
<td>8</td>
<td>10 ms</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>125</td>
<td>1, 2, 3, 4, 5, 6, 7, 8</td>
<td>10 ms</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>125</td>
<td>8</td>
<td>3 ms, 6 ms</td>
<td>3 ms</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>125</td>
<td>8</td>
<td>20 ms, 50 ms</td>
<td>20 ms</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>125</td>
<td>8</td>
<td>3-6 ms, 3-10 ms</td>
<td>3-6 ms</td>
</tr>
</tbody>
</table>

Table 5.2: List of tests performed on CAN test set-up.

In all the tests mentioned in Table 5.2, one node runs in read-only mode and logs all the messages going over the bus. So, the number of transmitting nodes is one less than what is mentioned in the column No. of nodes. The transmission of CAN messages is periodic for all the tests. The first set of tests focus on measuring the latency of CAN messages at different baud rates. The tests were run multiple times with different values of baud rates to observe the effect of baud rate on the latency. The second set of tests were performed to measure the latency of CAN messages for different size of data bytes inside the messages. Size of the data bytes inside CAN messages were increased from 1 to 8 bytes with an increment on 1 byte for each test. For the rest of the test cases, CAN baud rate was taken as 125 kbps and the number of data bytes in CAN message were taken as 8 in order to have a moderate value of the transmission time of CAN messages. Tests from number 3 onwards have two transmitting nodes and aims to observe the effect
of interference inside the CAN bus. Both the nodes have random start times. In test 3 and 4, the transmission of CAN messages has a fixed period while for test 5, it is uniformly distributed within an interval. The message having the high priority is indicated by its period in the Table 5.2. Test number 3 to 5 are a subset of the tests performed on the whole model. For measuring the latency of the CAN messages, the message generation time is recorded in the ECU inside Axiomtek before it starts writing the message on the bus and the end time is recorded in the ECU after it finishes the reading of the message. Hence, the latency observed is the combination of writing time (time taken by application to put message in the buffer of the driver), blocking time (in case the bus is not idle), transmission time, and reading time (time taken by application to read message from the buffer of the driver). The results of these tests are described below.

The average latency of the message to go from ECU 1 to ECU 2 through CAN bus at different baud rates is shown in Figure 5.3. The average latency decreases from 1.783 ms to 0.884 ms as the baud rate increases from 100 kbps to 500 kbps.

![Graph: CAN message latency at different baud rates](image)

**Figure 5.1:** CAN message latency at different baud rates.

As the latency is a combination of writing time, transmission time and reading time, the effect of the baud rate will be observed from the transmission time. The transmission time can be calculated from the formula: \( \text{number of bits/baud rate} \). For a CAN message of size 108 bits (8 bytes of data), the transmission time for 100 kbps baud rate will be 1.08 ms and for 500 kbps baud rate will be 0.216 ms. The difference of the transmission times at 100 and 500 kbps baud rate is 0.864 ms, which is very close to the difference of 0.899 ms of the two average latencies at 100 and 500 kbps baud rate. The impact of the size of the CAN message on the latency can be seen in Figure 5.2. It shows the average latency of CAN messages at a baud rate of 125 kbps having data bytes from 1 to 8 without facing any arbitration or delay from other CAN messages.

The average latency rise from 1.09 to 1.59 ms for 1 to 8 bytes of CAN data. For 1 byte of data (52 bits CAN message) at 125 kbps baud rate, the transmission time
5.2. MEASUREMENT OF THE LATENCY OF THE COMPONENTS

Figure 5.2: CAN message latency at different data sizes.

calculated from the formula will be 0.416 ms and 0.864 ms for 8 bytes of data. The difference of the transmission times of CAN messages having 1 and 8 data bytes is 0.448 ms. This difference is close to the difference of 0.50 ms observed in the average latencies of the CAN messages having 1 and 8 data bytes.

For populating the simulation CAN model, measurements from the test using 125 kbps baud rate, 8 bytes of CAN data transmitted from one node at a period of 10 ms were used. The transmission period was chosen large enough to have a smaller value of bus utilisation (to avoid any blocking from other messages). The frequency distribution of the latency for this test case in the test set-up is presented in Figure 5.3a.

![CAN message latency distribution](image)

(a)

![CAN message generation jitter](image)

(b)

Figure 5.3: Frequency distributions of latency of CAN messages (left) and jitter in the period of generation of CAN message (right) from a single-node CAN transmitter in the test set-up.

The frequency distribution in Figure 5.3a has a mean of 1.552 ms and a standard deviation of 0.0655 ms with a bus utilisation of 0.155. This frequency distribution was
obtained by first generating a histogram from the latency measurements using a bin size of 0.03 ms. The histogram was then normalised to obtain a frequency distribution. This frequency distribution was then provided as input to the CAN POOSL model. A small jitter was also observed in the periodic generation of CAN messages from the virtual ECU inside the Axiomtek. Average frequency distribution of those jitter readings from multiple tests is shown in Figure 5.3b. It means that for an ECU transmitting messages periodically at 10 ms interval, the time between two consecutive message transmission could be 9.98 and 10.04 ms. This jitter was provided to the POOSL model as well.

POOSL provides various standard distributions that can be used to model the distribution of latencies in the POOSL model. The standard distributions include normal, discrete, discrete uniform, gamma, exponential, Weibull, etc. To generate these distributions, one must supply the distribution parameters. POOSL then randomly selects a value from the generated distribution and uses it for computation. For example, for initialising normal distribution, mean and standard deviation are provided as input parameters. To generate a frequency distribution similar to the frequency distribution shown in Figure 5.3a, first, normal distribution was used. The discrete frequency distribution obtained from the normal distribution of the POOSL model is shown in Figure 5.4a. This frequency distribution has a mean of 1.552 ms and standard deviation of 0.0660 ms which is almost equal to the mean and standard deviation of the distribution obtained from the test set-up but the shape of the distribution is different. The difference between the shapes of the distribution can be observed by comparing Figures 5.4a and 5.3a.

To quantify the difference in the shapes of two discrete distributions, the following method is applied. Consider the labels on the x-axis of the graph of first distribution as \( \{x_1, x_2, \ldots, x_n\} \) and the values observed for the labels of x-axis on the y-axis as \( \{y_1, y_2, \ldots, y_n\} \). Similarly, for second distribution as \( \{X_1, X_2, \ldots, X_n\} \) and \( \{Y_1, Y_2, \ldots, Y_n\} \). For two distributions having the same set labels on the x-axis, the difference of the values of the distributions is calculated by the formula: 
\[
\sum_{i=1}^{n} \sqrt{(Y_i - y_i)^2}.
\]
The difference between the distributions shown in the Figures 5.4a and 5.3a is 0.194.

To obtain a smaller value of the difference between frequency distributions of test set-up and POOSL model, frequency distribution from the POOSL model was obtained using discrete distribution instead of normal distribution. POOSL generates a normal distribution with the input values of mean and standard deviation. For generating a discrete distribution, POOSL takes multiple inputs of the distribution values along with their weights. So, the bin values of the latency histogram of the test set-up along with their weight were given as input to the discrete distribution. To compare the frequency distribution obtained from POOSL with the frequency distribution obtained from the test set-up, \( t \)-test for two-sample assuming unequal variances was used. The bin size of the histogram of latencies obtained from test set-up was changed, and POOSL tests were run using histogram of different bin size as input to the discrete distribution until the \( t \)-test equates the two distributions with a minimal hypothesised mean difference.
5.2. Measurement of the Latency of the Components

Figure 5.4: Frequency distribution of CAN message latencies generated from normal (left) and discrete (right) distribution of POOSL model.

The final value of bin size chosen is 0.015 ms which produced a mean of 1.556 ms and standard deviation of 0.0648 ms with the distribution difference of 0.021 ms. The t-test passed with a hypothesized mean difference of 0.002 ms.

For this test on the POOSL model using discrete distribution, for a confidence level of 0.95, the confidence interval provided by POOSL is from 1.550 ms to 1.570 ms, and the minimum number of latency measurements required is 101. The results of the rest of the tests listed in Table 5.2 are presented in appendix B.

5.2.2 Test on Ethernet model

For measuring the time taken by an Ethernet message to go from the gateway to Axiomtek via Ethernet bus, multiple ping commands are sent from Linux operating systems in Axiomtek to the gateway device. The time taken by ping to return to the Axiomtek is stored and used for latency calculation. Half the time taken by ping command to return from the gateway is assumed to be the latency of Ethernet message to go from the gateway to Axiomtek. A frequency distribution of the measurement of latency of Ethernet message on the test set-up can be seen in the Figure 5.5. This frequency distribution has a mean of 0.142 ms and standard deviation of 0.0141 ms.

This frequency distribution is fed to the POOSL model in the same way as CAN distribution by using discrete distribution. The bin size used for creating the histogram from test set-up measurements is 0.005 ms. The frequency distribution obtained from the simulation model has a mean of 0.144 ms and standard deviation of 0.0142 ms. The difference in the frequency distribution is 0.058 and the t-test equates the two distributions with a hypothesised mean difference of 0.002 ms. For this frequency distribution, for a confidence level of 0.95, the confidence interval provided by POOSL is from 0.142 ms to 0.145 ms, and the minimum number of latency measurements required is 101.
CHAPTER 5. EXPERIMENTS AND RESULTS

5.2.3 Test on gateway

For obtaining the time the gateway takes to convert a CAN message to an Ethernet message, an experiment was performed on the test set-up (shown in Figure 5.6) in which ECU 1 transmits CAN messages of 8 data bytes at 125 kbps baud rate at an interval of 10 ms. No other node transmit any message. The message sent by ECU 1 is read by ECU 2 and by the WM ECU after conversion to Ethernet message.

To obtain a frequency distribution for the gateway conversion time, first, the time taken by messages to go from ECU 1 to the WM ECU (shown by red line in Figure 5.6) was measured. Second, from this measured time, the time taken in transmission via the CAN and the Ethernet bus (shown by dotted green line in Figure 5.6) was subtracted. This was done because the available gateway device does not provide any information about the timing of the messages received or sent by it. The measurement
of the Ethernet latency performed in the Section 5.2.2 was used. For measuring the time taken by a CAN message to reach the gateway from ECU 1, an assumption was made that the time taken by a message to reach the gateway from ECU 1 is half the time taken by a message to reach ECU 2 from ECU 1. The frequency distribution obtained from this test is shown in Figure 5.7.

![Gateway conversion time distribution](image)

Figure 5.7: Frequency distribution of message conversion times inside the gateway in the test set-up.

The frequency distribution shown in Figure 5.7 has a mean of 0.568 ms and a standard deviation of 0.049 ms. This distribution is given as input to the POOSL gateway model using discrete distribution as in previous sections. The bin size used for creating the histogram from test set-up measurements is 0.01 ms. The frequency distribution for conversion times of gateway generated by POOSL model has a mean of 0.572 ms and standard deviation of 0.050 ms. The difference in the frequency distribution is 0.3 and the t-test equates the two distributions with a hypothesised mean difference of 0.002 ms. For the distribution of gateway conversion times from POOSL, for a confidence level of 0.95, the confidence interval provided by POOSL is from 0.566 ms to 0.588 ms, and the minimum number of latency measurements required is 101.

5.2.4 Test on World model ECU

The behaviour of the WM ECU was explained in Section 4.2.3. In the WM ECU, the processing of a message task is characterized by its execution time, while the input-output (stimuli-response) delay of a message is characterized as latency. The value of execution time was measured from the test set-up and fed into the simulation model. The execution time of a message processing task inside the WM ECU is dependent on the hardware resources that it uses, the algorithm used inside the WM ECU, the number of messages received and the data in the messages that it receives. It was assumed that the execution time of a message processing task is not dependent on the above-mentioned factors. This assumption helps in providing a single frequency
distribution of execution times to the simulation model. A frequency distribution of the execution times of message processing tasks was measured via a specific test (test no. 1 in Table 5.1) and was considered to remain the same during other tests. The frequency distribution of the execution times of message processing tasks inside the WM ECU obtained from running test no. 1 in the Table 5.1 is shown in Figure 5.8.

![Figure 5.8: The frequency distribution of the execution times of message processing tasks inside the WM ECU.](image)

The distribution shown in Figure 5.8 has a mean of 0.653 ms and standard deviation of 0.022 ms. This distribution is given as input to the POOSL WM ECU using discrete distribution as in previous sections. The bin size used for creating the histogram from test set-up measurements is 0.01 ms. The distribution of execution time generated by the POOSL model has a mean of 0.657 ms and standard deviation of 0.022 ms. The difference in the distribution is 0.036 and the t-test equates the two distributions with a hypothesised mean difference of 0.004 ms. For this distribution, for a confidence level of 0.95, confidence interval provided by POOSL is from 0.653 ms to 0.66 ms, and the minimum number of latency measurements required is 101. With the help of the measurements of the execution times, POOSL model can determine the latency of the messages across the WM ECU.

The latency of messages inside the WM ECU is dependent on three factors, namely, message arrival time, execution time and message output time. As the output of the WM ECU is periodic (10 ms in our case), the latency of the message is mostly dependent on the arrival time. The message arrival time is dependent on the time of transmission of the message from the ECU connected to the CAN bus, jitter in the transmission of the message, and the time taken by a message to reach the WM ECU from the transmitting ECU. The path taken by a CAN message to reach the WM ECU from ECU 2 is shown by red colour in Figure 5.9.

For different values of period of transmission of CAN messages, considering the
5.2. MEASUREMENT OF THE LATENCY OF THE COMPONENTS

Figure 5.9: Simulation model with path from ECU 2 to the WM ECU marked by red line.

The period of arrival of messages at the WM ECU same as the period of transmission of CAN messages, i.e. neglecting the effect of jitter or latency of other components in the system, the arrival times and latencies of the messages can be observed from Figure 5.10.

A frequency distribution for the latency of messages across the WM ECU in the POOSL model for test no. 1 of table 5.1, with the inputs of latencies and jitter of CAN bus, Ethernet bus and gateway from the test set-up, is shown in Figure 5.11.

The distribution shown in the Figure 5.11 has a mean of 5.55 ms and a standard deviation of 2.868 ms. The minimum and maximum value of this distribution are 0.634 ms and 10.639 ms respectively. The reason why the latency distribution of the WM ECU is not a Gaussian-like distribution as compared to the latency distribution of other components of the system is a combination of the effect of jitter in the CAN transmission and periodic output of the WM ECU.

Considering the scenario of test number 1 in Table 5.1, the latency distribution of the messages for the path from ECU 2 to the arrival at the WM ECU (shown in Figure 5.9) is shown in Figure 5.12.

The distribution shown in Figure 5.12 includes the latency of the CAN, gateway and the Ethernet model. Because of this distribution, the arrival time of the messages at the WM ECU will vary. There is also a jitter in the period of generation of CAN messages from ECU 2 (as described in Section 5.2.1). The arrival time of the messages at the WM ECU is also affected by the jitter in the periodic generation of messages at ECU 2. For test number 1 on the simulation model, considering that the jitter has a mean
Figure 5.10: Arrival time and latency variation for different period of CAN message transmission.

Figure 5.11: Frequency distribution of the latencies of messages inside the WM ECU from the POOSL model.

value of zero, the distribution of latencies of messages inside the WM ECU is shown in Figure 5.13. The zero mean of the jitter will mean that the period of transmission of CAN message and the period of output message of the WM ECU will be same (10 ms) for test number 1.

The jitter in the periodic generation of CAN messages could also have a non-zero
5.2. MEASUREMENT OF THE LATENCY OF THE COMPONENTS

Figure 5.12: Frequency distribution of the latencies of the messages covering the path from CAN to Ethernet.

Figure 5.13: Frequency distribution of latencies of messages inside the WM ECU simulation model for equal period of message arrival and output.

The average value of the jitter observed in the period of generation or transmission (as no interference from other messages) of CAN messages is 0.01 ms, therefore, the average period becomes 10.01 ms instead of 10 ms. As the period of the CAN message transmission becomes more than the period of the output of the WM ECU (10 ms), the arrival time of messages at the WM ECU varies for consecutive messages. Figure 5.14 shows the effect of the jitter on the CAN transmission considering the average jitter of 0.01 ms for all the messages.

This jitter in the period of generation of CAN messages produces a shift in the transmission time of consecutive messages. This shift in the transmission of messages could go up to 1 ms after 100 messages. An example of the variation in the arrival time of the messages can be seen in the top part of Figure 5.15. As the output of the WM ECU is periodic, a change in the arrival time of a message will change its latency. The shift in the transmission of CAN messages from ECU 2, because of
CHAPTER 5. EXPERIMENTS AND RESULTS

Figure 5.14: CAN message transmission with and without the average jitter in the period of generation.

The jitter produces a shift in the arrival time of the messages at the WM ECU as well. As the arrival time increases, the latency of the message decreases. The shift in arrival time and the decrease in latency can be observed in the bottom part of Figure 5.15.

Figure 5.15: World model ECU timings with and without jitter in CAN transmission.

For test number 1 in the simulation model, the frequency distribution of the latency of first 100 messages inside the WM ECU because of the variation in the arrival time can be seen in top part of the Figure 5.16. The position of this frequency distribution on the time axis depends on the starting time of the message from ECU 2 and can vary for experiments having a different start time. The effect of the shift in the arrival time on the latency can be observed from the frequency distribution of the latency of the next 100 messages shown in the second part from the top in the figure 5.16. The next 100 messages will have a similar shift in the latency. The bottom part of Figure 5.16 depicts the frequency distribution of the latency of the WM ECU for 1000 messages.
5.3 Measurement of end-to-end latency

Tests listed in the Table 5.1 are performed on the simulation model and end-to-end latencies of the messages to go from ECU connected to the CAN bus to the output of the WM ECU is measured. Results of the frequency distributions obtained from the tests performed on the simulation model are listed in Table 5.3.

<table>
<thead>
<tr>
<th>Test no.</th>
<th>High priority</th>
<th>Low priority</th>
<th>Low priority</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean latency</td>
<td>Standard deviation</td>
<td>Min</td>
</tr>
<tr>
<td>1</td>
<td>7.059</td>
<td>2.870</td>
<td>2.120</td>
</tr>
<tr>
<td>2</td>
<td>7.186</td>
<td>2.889</td>
<td>2.100</td>
</tr>
<tr>
<td>3</td>
<td>7.210</td>
<td>2.997</td>
<td>2.070</td>
</tr>
<tr>
<td>4</td>
<td>7.133</td>
<td>2.881</td>
<td>2.100</td>
</tr>
</tbody>
</table>

Table 5.3: End-to-end latency results of the tests on the simulation model. All the values are in milliseconds.

Test results in the Table 5.3 provide separate results for high priority and low priority messages. Mean, standard deviation, minimum and maximum value of the
latency distributions are provided. This latency is a combination of the latency of CAN, gateway, Ethernet and the WM ECU simulation models. As can be observed from the results, the average latency is smaller for high priority messages than for low priority messages. Though, it might not be true in specific cases as both CAN and Ethernet have non-preemptive scheduling, so based on the start delay of the ECUs, the latency of low priority message could also be significant. The results of the same tests performed on the test set-up to validate the results of the simulation model are listed in Table 5.4.

<table>
<thead>
<tr>
<th>Test no.</th>
<th>Mean latency</th>
<th>Standard deviation</th>
<th>Min</th>
<th>Max</th>
<th>Mean latency</th>
<th>Standard deviation</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7.231</td>
<td>2.960</td>
<td>2.089</td>
<td>12.183</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>7.076</td>
<td>2.896</td>
<td>1.990</td>
<td>12.200</td>
<td>7.201</td>
<td>2.896</td>
<td>1.970</td>
<td>12.650</td>
</tr>
<tr>
<td>3</td>
<td>7.044</td>
<td>2.899</td>
<td>2.060</td>
<td>12.280</td>
<td>7.407</td>
<td>2.822</td>
<td>2.070</td>
<td>12.840</td>
</tr>
<tr>
<td>4</td>
<td>7.200</td>
<td>2.906</td>
<td>2.080</td>
<td>12.870</td>
<td>7.239</td>
<td>2.905</td>
<td>2.050</td>
<td>12.970</td>
</tr>
</tbody>
</table>

Table 5.4: End-to-end latency results of the tests on the test set-up. All the values are in milliseconds.

The test set-up results obtained also show a significant difference in the mean latency of high priority and low priority messages. The means and standard deviations of the test set-up and simulations results are very close. Table 5.5 provides the minimum number of measurements required in a test to obtain a confidence level and accuracy of 0.95. The confidence interval, for a confidence level of 0.95, obtained from the POOSL models and from the test set-up results for every tests are also mentioned in Table 5.5.

<table>
<thead>
<tr>
<th>Test no.</th>
<th>Min. no. of measurements</th>
<th>Confidence Interval (ms)</th>
<th>Confidence Interval (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Simulation</td>
<td>Test set-up</td>
</tr>
<tr>
<td>1</td>
<td>-</td>
<td>-</td>
<td>6.65 - 7.81</td>
</tr>
<tr>
<td>2</td>
<td>201</td>
<td>7.083 - 7.437</td>
<td>6.57 - 7.70</td>
</tr>
<tr>
<td>3</td>
<td>13701</td>
<td>6.812 - 7.488</td>
<td>6.64 - 7.76</td>
</tr>
<tr>
<td>4</td>
<td>401</td>
<td>6.806 - 7.472</td>
<td>6.65 - 7.79</td>
</tr>
</tbody>
</table>

Table 5.5: Confidence intervals for simulation and test set-up results.

The values of minimum number of measurements required to obtain a confidence level depend on the means of different batches of measurements. Batch size of 100 was used in the tests. In test number 2 and 4, period of CAN message transmission is less than the WM ECU output period. So, the WM ECU will receive more than 2 messages in a single output period and message latencies will have a small difference in the means of different batches. While for test number 1 and 3, CAN message transmission is equal or larger than the WM ECU output period. Hence, the WM ECU will receive almost one message in one output period. In such a scenario, the means of different batches will have a significant difference and will need more measurements to meet confidence level requirements. The POOSL model, for test number 1, could not run long enough to fulfil confidence level requirements and provide the values of minimum number of measurements and confidence interval. The graph of frequency distributions of end-to-end latencies of the first test for both the simulation model and test set-up...
5.3. MEASUREMENT OF END-TO-END LATENCY

can be seen in Figure 5.17.

![Figure 5.17: Frequency distributions of end-to-end latencies of test 1 for simulation model (left) and test set-up (right).](image)

The graph of the simulation model shown in Figure 5.17a is a combination of the distributions of the latency of the path from ECU 2 to the entry of the WM ECU shown in the Figure 5.12 and the distribution of the WM ECU latency shown in the Figure 5.11. The pie chart in Figure 5.18 depicts the percentage contribution of each of the component of the simulation model in the end-to-end latency in test number 1.

![Figure 5.18: Percentage contribution of the model components in the end-to-end latency distribution.](image)

As can be seen from the pie chart, though the WM ECU has the highest contribution in the end-to-end latency, still a change in the latency of CAN and gateway model produce a significant change in the end-to-end latency. The results obtained from the test set-up and simulation model (mentioned in Table 5.3, 5.4, 5.5) are similar to each
other in all the test cases. Hence, it can be concluded that the simulation-based approach can provide end-to-end latency measurements for the ADAS systems. The results of the end-to-end latency of rest of the tests on both the simulation model and ware models is described in appendix C.

5.4 Reflections on the results

Some of the reflections which can be made based on the results of the tests performed in this thesis are mentioned below.

1. The latency of a message through a communication bus increases with the increase in bus utilisation.

2. In the CAN bus, high priority messages have less latency compared to low priority messages, though, this may not be true in some situations.

3. A small jitter in the CAN message transmission can have a significant impact in the latency of an ECU with periodic output.

4. The latency distribution of an ECU with periodic output differs considerably from an ECU with an event-driven output.

Some conclusions on the results of the experiments performed in this thesis are mentioned in the next chapter.
Conclusions

This Chapter presents the conclusions of the work done in this thesis. Section 6.1 outlines the main work done in this thesis and puts the results in a broader context. Section 6.2 presents the main contributions of this thesis. In Section 6.3, some recommendations on how to design a system to make the system capable of performing timing analysis are made and Section 6.4 suggests some future work that could be done based on the work done in this thesis.

6.1 Summary

The core contribution of this thesis is presented in Chapters 2 to 5. Chapter 2 defined some timing metrics used for timing analysis. It also presented an overview of general automotive architecture along with the description of some of the components of the system. An automotive system named CACC was taken as an example. The working of the CACC system and its components were explained along with their timing behaviour. In the end, some challenges related to timing analysis in the automotive systems were outlined.

Chapter 3 provided a methodology to determine the end-to-end latency of different paths inside a system like CACC. The currently available methods for timing analysis can be broadly divided into two categories: analytical and simulation-based methods. Some examples of both these methods were provided. A small comparison of both the methods was done, and a selection was made based on the requirements of the CACC system. A simulation-based method was selected because of its ability to model complex interactions in the system.

Chapter 4 described an implementation of the selected methodology. Because of some limitation in the CACC system concerning timing analysis mentioned in chapter 4, a smaller version of the system was selected, and a hardware model was designed for this system. For this selected system, a simulation model was built in the POOSLIDE tool. The simulation model was made based on the principle that if the timing behaviour of the individual components of a system is known, then the timing behaviour of the whole system can be obtained by simulating the system captured in a POOSL model. Hence, the POOSL models of individual components of the system were built and then combined to make the whole model.

Chapter 5 described the tests performed on both the test set-up and simulation model. These results were then compared to validate the simulation model. The simulation model obtains the input-output latency of the individual components from the
test set-up and computes the complex interactions between and within the components. After obtaining the input-output latency of the different components, experiments to determine a frequency distribution of the end-to-end latencies of the simulation model were conducted. The results obtained from both the test set-up and simulation model were analysed, and it was concluded that a simulation-based approach could be used to determine end-to-end latency in the ADAS systems.

6.2 Contributions

In this section, the main contributions of this thesis are listed to confirm if the goals mentioned in Chapter 1 are achieved or not. The results of the POOSL model confirm that a simulation-based approach can provide a solution to the problem statement introduced in Chapter 1. Main contributions of this thesis are as follows:

- A small comparison of simulation-based and analytical methods was done based on the ability to cover the corner cases, scalability and ability to model complex interactions (see Section 3.3 for more details).

- A POOSL model capable of determining the distribution of end-to-end latency of the selected system is designed.

- The designed POOSL model consists of a heterogeneous communication bus and a special ECU like the WM ECU. These are the most conceptually important components for an automotive architecture and cover all the types of behaviours found in a real system.

- The POOSL model provides timing information at the component level as well as at the system level and can measure the effect of various timing parameters like jitter, the priority of the message, variation in the time interval of message generation and so on.

- The automotive components can be added or removed from the POOSL model to design other automotive systems.

- Implemented confidence level analysis in the simulation model which can provide the number of measurements required to obtain a certain confidence level in the result and also provide a confidence interval for the observed value.

6.3 Recommendations

Some recommendations that can be provided based on the work done in this thesis to facilitate validation of the simulation model are as follows:

- Use a common clock for all the components of the test set-up so that time taken by a message to reach one component from another can be measured.
• Put the time of message creation or transmission in the message sent from ECUs wherever possible to get information about the time taken in message transmission. Using the same time of message creation of source ECU in the messages sent from consecutive ECUs can easily provide end-to-end latency.

• For every ECU, log the messages sent and received along with the timestamps to obtain the latency of messages inside every ECU.

• In the test set-up, avoid equipment which does not have the capability of providing the timestamps and logging the messages, like the gateway device used in the thesis. It limits the timing analysis capability of the system.

• Implement the system in such a way that it has least possible amount of jitter in the system components. The jitter can have a significant impact in the latency of messages especially for an ECU having periodic output.

6.4 Future work

This thesis provides a basis on which some work can be done in the future. Some suggestions for the future work are mentioned below:

• Modify the POOSL model to be able to measure the statistics of message loss in the communication buses.

• Perform tests on the Ethernet model by connecting multiple nodes (more Axiomteks or gateways) to the bus on the test set-up and then on the POOSL model.

• Extend the POOSL model to test the complete CACC system.

• Integrate the simulation model into a higher level tool like enterprise architect from which the simulation model can be instantiated.
Bibliography


This appendix gives an overview of the syntax and working of the POOSL language. The language is structured into three layers. The first layer, data layer, captures passive data objects. The second layer, process layer, active process objects and the third layer, architecture layer, describes system structure and architecture.

All data in a POOSL models is represented by data objects, which are instances of data classes. A data class has a name, a (single) inheritance relation, instance variables and instance methods. The behaviour of data is sequential and defined by its instance methods or (data) methods. Methods are similar to functions used in Java or C++. The behaviour of ordinary data methods is specified by expressions shown in figure A.1.

![Figure A.1: POOSL data expression, taken from [6].](image)

To model the basic components, POOSL provides process objects or processes, which are instances of process classes. Their behaviour is described in the process layer. Defining a process class involves specifying a name, instantiation parameters and instance variables, a port interface and message interface, instance methods, an initial method call and an optional single inheritance relation. The instantiation parameters and instance variables are the attributes of a process object. Processes may communicate their encapsulated data objects via ports. The port interface lists the names of the ports via which instances of a process class may communicate messages. The message interface lists the signatures of all possible messages and includes for each message the port name, the symbol ! or ? for message send and receive respectively, the message name and a list of types of the message parameters. The behaviour of
processes is defined by the instance methods or process methods of the corresponding process class. The starting behaviour of a process is defined by the initial method call. The body of a process method is defined using the statements shown in figure A.2.

The statements in figure A.2 provides the syntax of how different statements can run either in parallel or in sequence. Other useful syntax like running a statement conditionally are also mentioned. Most data expressions of are also valid inside process methods.

The architecture layer allows to hierarchically build a structure of process objects connected with channels, using clusters. Clusters are defined by cluster classes. A cluster class has a name, instantiation parameters, ports, a message interface and a behaviour specification. Channels may connect only instances inside the cluster, but channels may also connect to the outside ports of the cluster. Next to defining cluster classes, the architecture layer defines the system specification of a model. The system specification includes a behaviour specification and a list of all classes defined for a model. This behaviour specification defines how processes and clusters are interconnected at the highest hierarchical level.
CAN model test results

The results of the last three tests mentioned in table 5.2, performed on the CAN simulation model, are presented in the table B.1.

<table>
<thead>
<tr>
<th>Test no.</th>
<th>High priority</th>
<th>Low priority</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean latency (ms)</td>
<td>Standard deviation (ms)</td>
</tr>
<tr>
<td>3</td>
<td>1.685</td>
<td>0.308</td>
</tr>
<tr>
<td>4</td>
<td>1.573</td>
<td>0.127</td>
</tr>
<tr>
<td>5</td>
<td>1.700</td>
<td>0.337</td>
</tr>
</tbody>
</table>

Table B.1: Results of tests performed on the CAN simulation model.

The values in table B.1 shows the mean and standard deviation of the latency values provided by the simulation model. The tests results shows the effect of priority in CAN messages. Effect of bus utilisation on the average latency can also be observed. To verify the results of the simulation model, same tests were performed on the test set-up. The results of the tests on the test set-up are given in table B.2.

<table>
<thead>
<tr>
<th>Test no.</th>
<th>High priority</th>
<th>Low priority</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean latency (ms)</td>
<td>Standard deviation (ms)</td>
</tr>
<tr>
<td>3</td>
<td>1.609</td>
<td>0.223</td>
</tr>
<tr>
<td>4</td>
<td>1.561</td>
<td>0.101</td>
</tr>
<tr>
<td>5</td>
<td>1.610</td>
<td>0.204</td>
</tr>
</tbody>
</table>

Table B.2: Results of tests performed on the CAN test set-up.

The results obtained from the test set-up are similar to the results of the simulation model.
The graphs of the test results mentioned in Table 5.3 and 5.4 are shown in this appendix. For test 1 in Table 5.1, considering that the jitter in the periodic transmission of CAN message has zero mean, the frequency distribution of the end-to-end latency of the simulation model is shown in Figure C.1. The end-to-end latency frequency distribution obtained from non-zero mean was shown in Figure 5.17.

**Figure C.1:** Frequency distribution of end-to-end latencies of messages in test 1 on POOSL model with jitter having zero mean.

For test 2 in Table 5.1, considering that the jitter in the periodic transmission of CAN message has zero mean, the frequency distributions of the end-to-end latencies of both the high and low priority messages in the simulation model are shown in Figure C.2.

**Figure C.2:** Frequency distribution of end-to-end latencies of high priority (left) and low priority (right) messages in test 2 on POOSL model with jitter having zero mean.
The frequency distributions of end-to-end latencies of high priority messages in test 2 for both the POOSL model and test set-up, having non-zero mean, are shown in Figure C.3.

![Figure C.3: Frequency distribution of end-to-end latencies of the messages in test 2 on simulation model (left) and test set-up (right).](image)

A shift in the transmission of CAN message is not produced in the results shown in the Figure C.2, as in the case of periodic transmission having non-zero mean (0.01 ms in our case) in the results shown in the Figure C.3.

Similarly, for test 3 in Table 5.1, considering that the jitter in the periodic transmission of CAN message has zero mean, the frequency distributions of the end-to-end latencies of both the high and low priority messages in the simulation model are shown in Figure C.4.

![Figure C.4: Frequency distribution of end-to-end latencies of high priority (left) and low priority (right) messages in test 3 on POOSL model with jitter having zero mean.](image)

The frequency distributions of end-to-end latencies of high priority messages in test 3 for both the POOSL model and test set-up, having non-zero mean, are shown in
Similarly, for test 4 in Table 5.1, considering that the jitter in the periodic transmission of CAN message has zero mean, the frequency distributions of the end-to-end latencies of both the high and low priority messages in the simulation model are shown in Figure C.6.

Figure C.6: Frequency distribution of end-to-end latencies of high priority (left) and low priority (right) messages in test 4 on POOSL model with jitter having zero mean.

The frequency distributions of end-to-end latencies of high priority messages in test 4 for both the POOSL model and test set-up, having non-zero mean, are shown in Figure C.7.
Figure C.7: Frequency distribution of end-to-end latencies of messages in test 4 on simulation model (left) and test set-up (right).