Universidade de Aveiro Departamento de
Electrónica, Telecomunicações e Informática,
2012

**Rafael José
Antunes Pinto**

DroidShark: Sistema de Telemetria Android para o veículo HammerShark

DroidShark: Android Telemetry System for the HammerShark vehicle

Universidade de Aveiro Departamento de
Electrónica, Telecomunicações e Informática,
2012

Rafael José Antunes Pinto

DroidShark: Sistema de Telemetria Android para o veículo HammerShark

DroidShark: Android Telemetry System for the HammerShark vehicle

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Prof. Dr. José Maria Amaral Fernandes, Professor Auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro e do Prof. Dr.Manuel Bernardo Salvador Cunha, Professor Auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro.

Dedico este trabalho aos meus pais e à minha irmã.

I dedicate this work to my parents and sister.

o júri / the jury

presidente / president	**Prof. Doutor Joaquim Manuel Henriques de Sousa Pinto** Professor Auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro
vogais / examiners committee	**Prof^{a}. Doutora Ana Cristina Costa Aguiar** Professora Auxiliar do Departamento de Engenharia Electrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto
	Prof. Doutor José Maria Amaral Fernandes Professor Auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro (orientador)
	Prof. Doutor Manuel Bernardo Salvador Cunha Professor Auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro (co-orientador)

agradecimentos / acknowledgements

Este trabalho serviu para solidificar os meus conhecimentos na área da computação móvel durante o último ano que passou. O meu obrigado aos meus orientadores, o Professor Doutor Manuel Bernardo Salvador Cunha, pelas sugestões e confiança depositada em mim. Ao Professor Doutor José Maria Amaral Fernandes, pela paciência e disponibilidade que sempre mostrou ao longo deste trabalho.
Ao meu pai, José António e à minha mãe Maria Augusta agradeço todo o apoio incondicional durante estes anos, sem eles não estaria aqui hoje. À minha irmã Gisela pela ajuda que me proporcionou durante este último ano.
Um grande obrigado a todos os meus amigos, pela companhia e conselhos ao longo deste trabalho.

palavras- chave Android, Sistema de Telemetria, Shell Eco- Marathon

Resumo

HammerShark é um veículo resultante de um projeto pluridisciplinar na Universidade de Aveiro, envolvendo várias áreas tais como mecânica, electrónica e informática. O veículo foi desenvolvido com o objectivo de participar na Shell Eco- Marathon, uma prova académica com o intuito de desenvolver e testar veículos ecológicos. O vencedore é a equipa que percorre a maior distância, utilizando a menor quantidade de energia.
O HammerShark introduziu algumas inovações, tais como o *CAN bus* com um interface Bluetooth, permitindo que todos os parâmetros do veículo e da prova possam ser obtidos através deste. Em todas as provas motorizadas, o sistema da telemetria é uma parte fulcral para atingir o sucesso no evento.
Com o aparecimento de dispositivos móveis, que utilizam sistemas operativos *open- source*, abriu-se a possibilidade de utilizar este tipo de dispositivos como meio de suporte aos sistemas de telemetria.
Nesta dissertação propomos um novo sistema de Telemetria – DroidShark, para o HammerShark - baseado no *Android OS*. Com o suporte por parte do Android para comunicação inter- processos entre aplicações *third- party*, foi possível desenvolver uma arquitetura modular capaz de concretizar todos os objectivos propostos.
O DroidShark é composto por duas unidades – car unit e pit unit -, onde o car unit comunica com o HammerShark e é responsável por providenciar todos os dados recolhidos à pit unit, por forma a informar todos os restantes membros da equipa que se encontram na *pit lane*.
O DroidShark demonstra que a incorporação de dispositivos móveis em sistemas de telemetria é uma solução com grande potencial.

Keywords Android, Telemetry System, Shell Eco -Marathon

Abstract

HammerShark is a vehicle, resulting from a multidisciplinar project from University of Aveiro involving work from several areas such as mechanics, electronics and informatics. The vehicle was build with the aim to participate in the Shell- Eco Marathon, a challenge for college students to design build and test energy efficient vehicles. The winner is the team that covers the farthest distance using the least amount of energy.
HammerShark introduces some innovations, namely the CAN bus with a Bluetooth Interface for publishing data through it.
On every race challenges, the telemetry solution is a fundamental part in order to achieve success in the event. With the introduction of the CAN bus, a new and improved telemetry system could be developed to assist the decision of the driver and/or the crew during and after the race.
With the emergence of mobile devices, using open –source operating systems, a new door was open to develop new telemetry systems based on these kind of devices.
In this dissertation we propose a new telemetry system - DroidShark, for the HammerShark vehicle - based on Android OS. Since Android OS supports inter- process communication between third- party applications, it was possible to develop a modular architecture to achieve all proposed goals.
The DroidShark is composed by two units – car unit and pit unit -, where the car unit communicates with HammerShark and is responsible to provide all acquired information to the pit unit, in order to inform all remaining team members that stand at the pit lane. DroidShark demonstrates that incorporating mobile devices in telemetry solutions have a significant potential.

Contents

List of Figures

List of Tables

List of Abbreviations

AP	Access Point
API	Application Programming Interface
CAN	Controller Area Network
CORBA	Common Object Request Broker Architecture
DTU	Danmarks Tekniske Universitet
ECU	Engine Control Unit
GPS	Global Position System
GSM	Global System for Mobile Communications
ICEG	Internal Combustion Engine Group
IDL	Interface definition language
IPC	Inter process communication
IrDA	Infrared Data Association
JVM	Java Virtual Machine
KML	Keyhole Markup Language
LTE	Long Term Evolution
MEK	Mekanik (DTU Mekanik)
NFC	Near Field Communication
OS	Operating System
PDA	Personal Digital Assistant

PMD	Personal Measurement Device
QoS	Quality of Service
RF	Radio Frequency
RFID	Radio Frequency Identification
RMI	Remote Method Invocation
RPC	Remote Procedure Call
SDK	Software Development Kit
TCP	Transfer Control Protocol
UDP	User Datagram Protocol
UMTS	Universal Mobile Telecommunications System
URI	Uniform Resource Identifier
USART	Universal Synchronous and Asynchronous serial Receiver and Transmitter
USB	Universal Serial Bus
WLan	Wireless Local Area Network

Chapter 1

Introduction

The Shell Eco- Marathon [1] is an annual event that takes place in America, Europe and Asia, where high school and college students around the world compete against each other. The winner is the team that go the farthest distance using the least amount of energy and has achieved all minimum goals imposed by the organization of the event (e.g. Number of laps, Minimum average speed, Trial time). To accomplish the goals, a good telemetry solution is important in order to inform the driver how he is doing during the race and for post-race analyses by the team. It is also used to gain knowledge for future trials.

University of Aveiro had its first participation at the Shell Eco- Marathon in 1997 with Icaro [2], a multidisciplinary project involving work from different areas, namely mechanics, electronics and informatics. Icaro introduced some concepts such as telemetry solutions supported on a PDA and customized hardware interfaces. The telemetry system allows the team to analyze the acquired information after the race and support a new approach to achieve lower consumptions.

Based on the Icaro experience, a new car has been developed. Its shape was inspired by the hammerhead shark and therefore named HammerShark. HammerShark was conceived and implemented from beginning with a CAN bus architecture that allows both control and monitoring of the mechanics and electronics of the car. The central node of the telemetry system was also the PDA, which was on the steering wheel and works like a gateway using Bluetooth to establish the connection and retrieve all data provided by the car.

With the emergence of smartphones with open operating systems like Android OS [3], it was only a question of time to devise a new monitoring/telemetry solution for the HammerShark vehicle based on them.

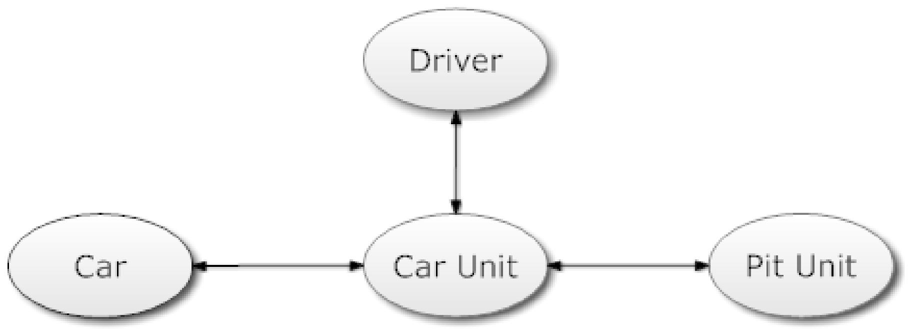


Figure 1.1: Overall abstract schema

1.1 Objectives

The objective of this work is to propose a new telemetry system based on Android mobile phone to replace the existing telemetry system in HammerShark for using at the Shell Eco – Marathon challenge, being the focus of this work the telemetry system components and the capabilities of the Android OS.

The new telemetry system is centred on a smartphone that is inside the car –named car unit, which is responsible to interact with three entities (Figure 1.1).

- Car – A communication with the vehicle must be established in order to obtain the telemetry data from it.
- Driver – The driver must know how he/she is doing during the race, and therefore he must be provided with information (e.g. speed, rpm, lap number, time elapsed).
- Pit Unit – The remaining team members stay at the pit lane during the race and in order to follow live the course of the race , this unit must be provided with the telemetry data acquired from the car.

1.2 Dissertation structure

This dissertation is divided into the following chapters:

In **Chapter 1** (the current one) we present the main motivation for this dissertation and enumerate the dissertation objectives and main contributions.

In **Chapter 2** entitled "Telemetry System", we will do a short review on existing telemetry systems that can be used as motivation for our work and present a comparison between these telemetry systems and our telemetry system. On this chapter we also present several standards that can be used by the car unit to establishing a connection with the car and with the pit unit.

In **Chapter 3** we propose a modular architecture (DroidShark) for our telemetry system and present the scenario where our system will be inserted.

In **Chapter 4** we present the implementation of our system and review IPC mechanisms provided by the Android OS.

In **Chapter 5** we evaluate the performance of two IPC mechanisms provided by the Android OS. The evaluation supports our choice and permits the analysis of the behaviour of each mechanism in this scenario.

In **Chapter 6**, “Conclusions and future work”, discusses the accomplishments during the development of this work.

Chapter 2

Telemetry systems

In this section we will address the state of the art in car telemetry system and we will address the basic concept of Telemetry. There are several perspectives that we must consider when implementing a telemetry system, namely the system and the communication.

Telemetry systems play an important role in motor sports. These systems provide the necessary information to the engineers, in order to adjust the car and detect failures before they occur and try to take an action in order to minimize these. To provide the engineers with the acquired data, the communication between the car and the engineers plays an important role, since data loss is critical in a telemetry system. There are two important aspects when we refer the communication link of a telemetry system: the maximum range of the communication standard and the bandwidth for data transmission.

Beside the communication link, the telemetry system itself must be reliable. There are several solutions to implement a telemetry system. With the appearance of open operating system based smartphones, they become an alternative to embedded telemetry systems. With the use of a smartphone, there is no longer the need of using custom hardware to provide communication to a remote device and it can be used as a display for the driver.

Further in this section we will review telemetry systems, which are based on embedded systems and on mobile operating systems.

2.1 Telemetry

The word telemetry is derived from two Greek words *tele* (remote) and *metron* (measure). The term can be applied to all systems that are measured from a remote distance. Therefore, to have a telemetry system, it is necessary to have one or more transmitters and one or more receivers. The link between the transmitter and the receiver is usually created with wireless communication but it can be created as well via a telephone line or some other communication medium.

Telemetry systems are very important in the motor sport field. They are used in all categories and types of motor sports, ranging from motorcycles to rally cars. In this field, there are telemetry systems working in every second that the vehicle is operating. The goal of telemetry in the motor competition world is to measure all the necessary parameters to create a real vision of the systems so that engineers could predict the car's behaviour in the future and therefore prevent accidents and major problems. Moto GP and Formula 1 are two motor sport categories, where the pilots are constantly at risk and the prediction of failures could be crucial to protect the driver.

The emergence of open operating systems for mobile devices, that become more and more powerful, give us the opportunity to use a modern smartphone to monitor the vehicle from a remote location. With the use of a smartphone, the need of custom hardware for obtaining the measures becomes optional. We can use the internal sensors and modules to retrieve all measures from the car and forward this information to any location. However, there are concerns about the communication standard that we will use to provide live monitoring at the remote location.

2.2 Car telemetry systems

This dissertation is focused on building a telemetry system based on an Android operating system based smartphone for the HammerShark vehicle. In order to better understand which telemetry systems already exist that have similarities to the one proposed here, we present a state of the art review on car telemetry systems. Our review was supported on a web and publications research based on the keywords Android, Telemetry System and Shell Eco-Marathon.

Our focus was to find out which projects/products were commercial or the result of a research by a university or institute. Since our telemetry system is mobile device based, we focused mainly on systems that were also based on mobile operating systems. However, telemetry systems that participate at the Shell Eco- Marathon, which are based on embedded systems were also considered in our review.

With the projects/products identified, we focus on comparing the architecture of these projects with our proposal and the communication technology that is used to provide data to a remote location. The software used to analyze the telemetry data and the ability to offer live monitoring during the race, was also a subject of analysis.

We have identified five telemetry systems that fit in our requirements. Two telemetry systems have commercial origin, while the other three were the result of a research work done by universities. Torque Pro [5] and AMG Performance Media [6] are two commercial telemetry systems based on the Android operating system. Torque Pro is available at Google Play [4], AMG Performance Media can be ordered as an extra on all AMG models of the Mercedes- Benz vehicles.

DTU Innovator [9], Pingu II [10] and Agilis Eco Car [11], are three telemetry systems produced in universities. DTU Innovator is a result of a research at the Technical University of Denmark, while Pingu II was produced at the Hamburg University of Applied Sciences (HAW) in Germany. Agilis Eco Car was developed by the University of Pontificia Comillas in collaboration with the Royal Institute of Technology from Stockholm.

A detailed comparison of the review is summarized in table 2.1.

	Torque Pro	AMG Performance Media	DTU Roadrunners	Pingu II	Agilis Car	DroidShark
Commercial	YES	YES	NO	NO	NO	NO
Device	Mobile device	Embedded system	Embedded system	Embedded system	Embedded system	Mobile device
Communication with the car	YES (Bluetooth)	YES (N/A)[1]	YES (N/A)[1]	YES (USB)	YES (Integrated and RS- 232)	YES (Bluetooth)
Information to the driver	YES	YES	NO	NO	NO	YES
Live monitoring	YES (GSM/UMTS)	NO	YES (GSM)	YES (Wi- fi)	YES (Transmitter – Receiver chips)	YES (Wi- fi)
Data Logging	YES (internal storage)	YES (USB- stick)	YES (internal storage)	YES (USB- stick)	YES (internal storage)	YES (internal storage)
Post analysis	YES (Web viewer)	YES (N/A)[1]	YES (Custom software)[2]	YES (Custom software)[2]	YES (Custom software)[2]	YES (Tablet)[3]

Table 2.1: Summarized comparison of the telemety systems

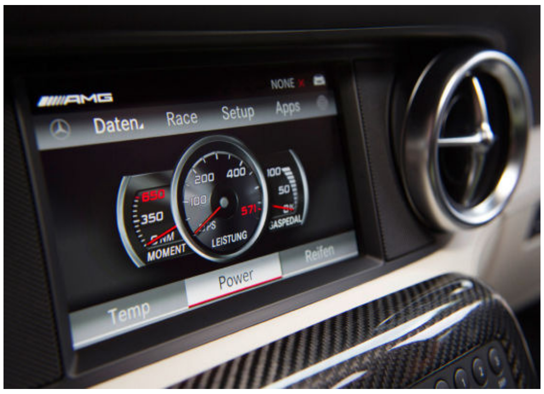
[1]N/A – Information not available
[2]Custom Software – Software developed for this purpose by the institute/team
[3]Tablet – Option available at the tablet application

2.2.1 Torque Pro

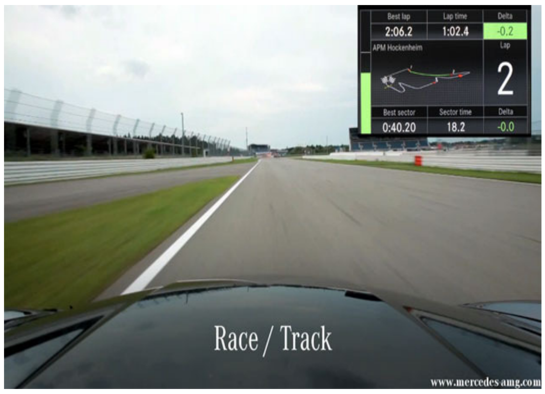
Torque Pro [5] is one application developed for Android OS based devices that serves as a telmetry system for regular vehicles. With an average rating of 4.8 and with until this instant 7'740 ratings, Torque Pro is a popular application at Google Play store [4]. The application get On- Board-Diagnose (OBD) fault codes, measures the performance, sensor data and allow the user to view engine data in Google Earth. This system has also the ability to upload engine data in real time to a personal webserver or to the web viewer created for this purpose by the developers, at the product web page. The upload of the engine data, provides the ability for the user to analyze that data whenever he wants. The data is acquired over Bluetooth, which means that the user must buy separately a Bluetooth adapter for the OBD diagnostic connector. The need of an OBD diagnostic connector is a requirement for using this application, which could be a problem for older vehicles, since they may not be fitted with such connector.

2.2.2 AMG Performance Media

With the emergence of open mobile operating systems, the car industry gained interest of using such operating systems to fit their needs. Mercedes –Benz has followed the trend of using mobile operating systems and developed the AMG Performance Media [6]. The AMG Performance Median (Figure 2.1(a)) uses the Android operating system and combines numerous telemetric displays such as various engine data, lateral and linear acceleration, lap times with high –speed mobile Internet connectivity. The system is activated by pressing an AMG button in the car, making all functions visible on a high- resolution colour display that is also used for the standard multimedia system. Beside the telemetry data enumerated above, the system provides a "Track" menu (Figure 2.1(b)). This mode offers various recording possibilities, for example individual lap times on a closed race circuit, but also sector times including an analysis and memory function. This mode has the goal to help the driver to make continuous improvements to the personal performance. All data can be saved on a USB- stick and analyzed on the PC at home. In addition to all the telemetric displays, AMG Performance Media provides fully- fledged, mobile high- speed internet access. The driver is also able to install apps, which he has downloaded from the internet via the system browser or alternatively using a USB stick that can be connected. This systems provides overall similar functions to those of a smartphone.



(a) AMG Performance Media in a car[7]



(b) Track Mode of the AMG system[8]

Figure 2.1: AMG Performance Media

2.2.3 DTU Roadrunners

The DTU Roadrunners [9] is a team from the Technical University of Denmark, who made his first participation in the Shell Eco- Marathon in January 2004 as a result of an invitation from Shell Denmark. The ICEG at MEK coordinated the start- up and rapidly 20 students enrolled to the project. They claim that to be in front of the Shell Eco- Marathon, the car must be optimized and the team must have the ability of learning from past mistakes. Therefore they invested on the data acquisition part, because it is an essential part of the whole optimization process. As a result of all effort spent, the DTU team finished 5th in the first participation at the Shell Eco- Marathon. They achieve a result of 583km/l in the alternative fuels category.

The Innovator – car of the DTU Roadrunners, owns a system that is monitoring data, such as the current voltage, gps position, pressure, and temperature. The data acquisition system in the Innovator is integrated in the main system. The data is transferred via a GSM modem during a run, thus making it possible to see the stats in real time in the pit. If any problems are encountered during a run, the driver is made aware and the faults tried to be corrected. The data is viewed and logged in a program (Figure 2.2) made for the Innovator for later analysis. On the program is possible to see where the optimization process is needed.

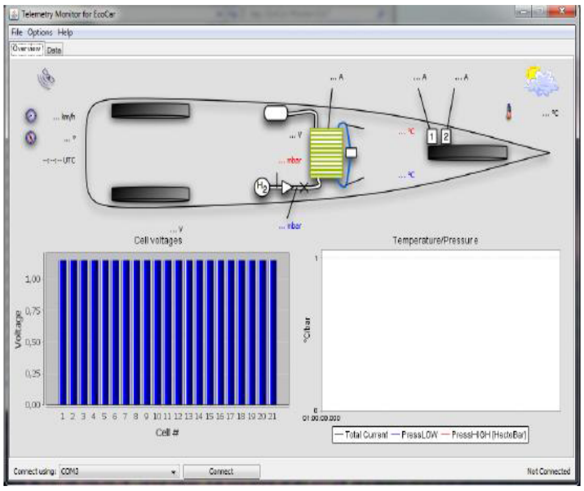


Figure 2.2: Telemetry application of the DTU Innovator[9]

2.2.4 Pingu II

"Pingu II" is a car developed at the Hamburg University of applied sciences, with a custom Telemetry System (Figure 2.3) [10].

The core of the whole Telemetry System of "Pingu II" is the "Telemetriebox", which is installed in the car and fully developed at the Hamburg University. The box is made of transparent polycarbonates, in order to accomplish the rule of the Eco- Marathon. The rule says that all technology installed in the vehicle must be visible. The box consists of the following components:

- Central board sensors, where the sensors are connected and converted the input values for the signal pickup.
- Serve power for all devices at the system.
- A PMD (Personal Measurement Device), which transmits directly the values of the USB bus to the system. Over the USB bus its possible to connect easily additional PMD's or USB devices.
- A communication board, to be possible to communicate with the box without connecting a laptop.
- USB Hub, to provide distribution for the internal and external devices.

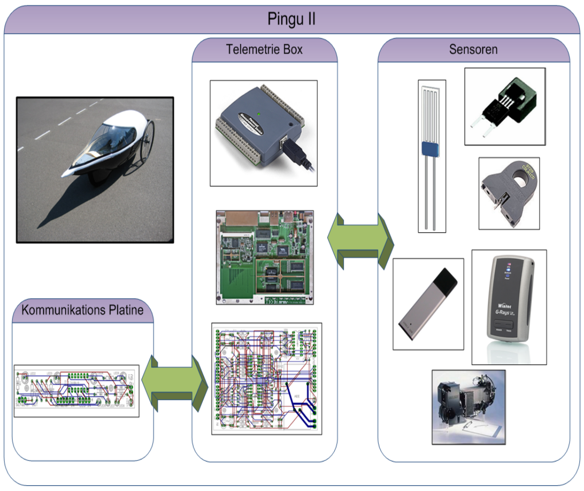
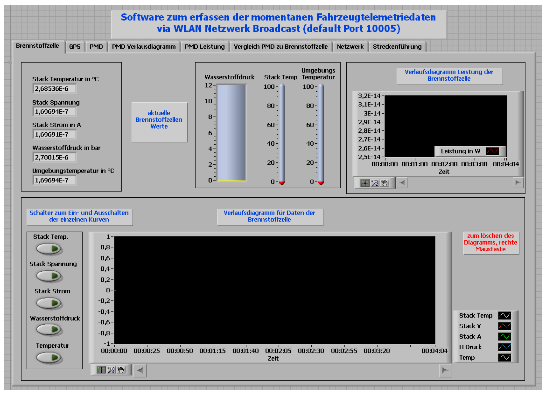


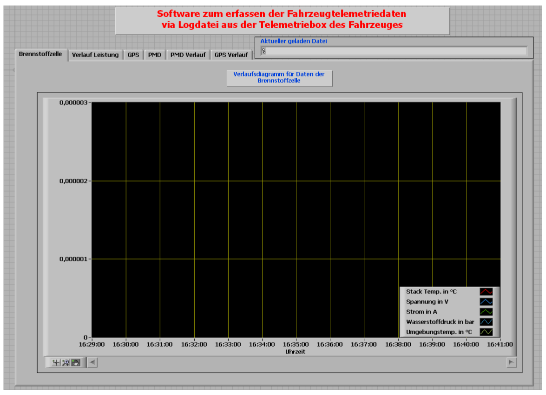
Figure 2.3: Telemetry Solution of the Pingu II vehicle[10]

On one of the USB interface available of the box, there is a USB- stick attached during the race, where all collected data are stored. The box has also been fit with a Wi-fi module, which broadcasts data using the UDP protocol to a laptop that stays at the pit. Since Wi-fi has a limited range, the data are only sent if the vehicle is in the range of the network. For the analysis of the data collected during a race, an application has been developed. The program offers the ability to analyze the live data received from the car (Figure 2.4(b)) or to analyze the collected data during the race offline, by loading the file saved at the USB-stick(Figure 2.4(a)).


Software zum erfassen der momentanen Fahrzeugtelemetriedaten via WLAN Netzwerk Broadcast (default Port 10005)
Brennstoffzelle
GPS
PMD
PMD Verlausdiagramm
PMD Leistung
Vergleich PMD zu Brennstoffzelle
Netzwerk
Streckenführung
aktuelle Brennstoffzellen Werte
Wasserstoffdruck
Stack Temp
Umgebungs Temperatur
Verlaufsdiagramm Leistung der Brennstoffzelle
Leistung in W
Zeit
Schalter zum Ein- und Ausschalten der einzelnen Kurven
Verlaufsdiagramm für Daten der Brennstoffzelle
zum löschen des Diagramms, rechte Maustaste
Stack Temp.
Stack Spannung
Stack Strom
Wasserstoffdruck
Temperatur
Stack Temp
Stack V
Stack A
H Druck
Temp

Stack Temperatur in °C 2,68536E-6
Stack Spannung 1,69694E-7
Stack Strom in A 1,69691E-7
Wasserstoffdruck in bar 2,70015E-6
Umgebungstemperatur in °C 1,69694E-7

(a) Pingu II post analysis application[10]



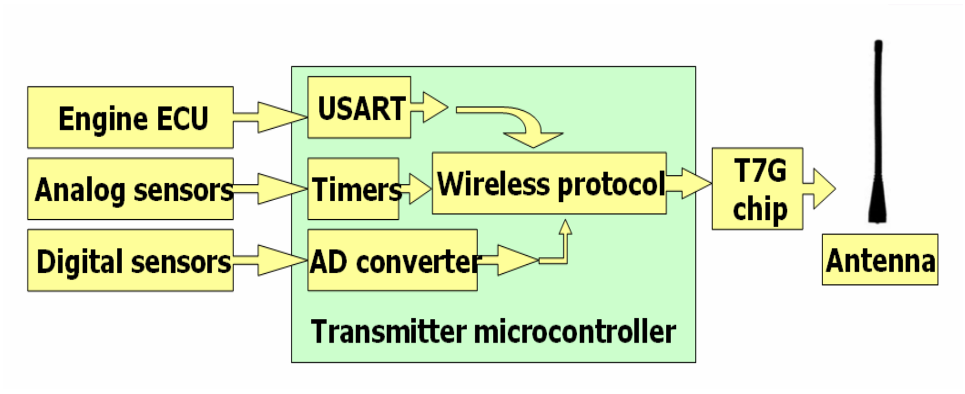
(b) Pingu II live analysis application[10]

Figure 2.4: Pingu II analysis application

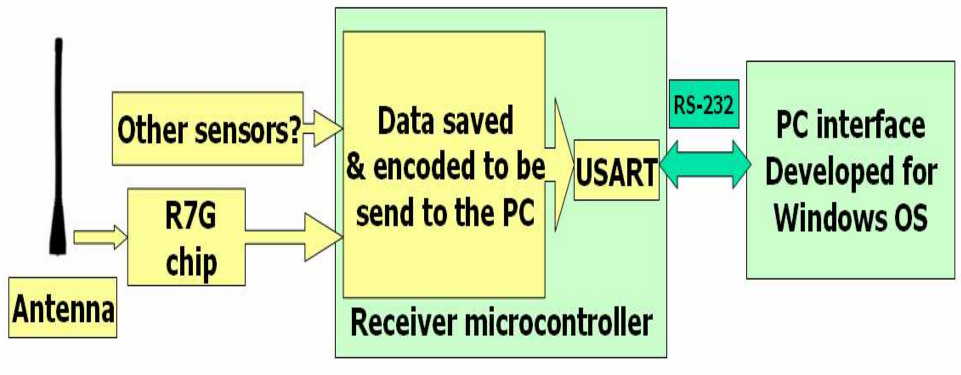
2.2.5 Agilis Eco Car

Agilis Eco car [11] was a large project of the University of Pontificia Comillas, but the project was organized and planned in the Mechatronics Laboratory of the Machine Design Department of the Royal Institute of Technology in Stockholm. The aim was the participation in the Shell Eco- Marathon.

The system of the Agilis Eco Car consists in two blocks, the transmitter block (Figure 2.5(a)) and the receiver- PC block (Figure 2.5(b)). The transmitter block comprises a microcontroller that has three different ways to receive data externally. It can receive data from analog or digital sensors, which were connected directly to the microcontroller's port and it can receive data from the other controllers of the car through the serial port. Once the data, from the sensors and from the microcontroller, arrives to the emitter board, they have to be saved in its microcontroller. All data are saved together in an array, which can save 64 different values where the order inside the array represents the identifier field. The value is saved in the array position.



(a) Transmitter block diagram[11]


Antenna
Other sensors?
R7G chip
Data saved & encoded to be send to the PC
USART
Receiver microcontroller
RS-232
PC interface Developed for Windows OS

(b) Receiver-PC block diagram[11]

Figure 2.5: Agilis Eco car system

The communication between the transmitter and the receiver is unidirectional and the transmitting protocol sends the data constantly. By sending constantly, it means that the first packet sent will be the identifier number one and its value; the second packet will be the identifier number two and its value. When the last packet has been sent- it is the one with the highest identifier- the first value is sent again, even if a new measure has not happened. The process is graphically depicted at figure 2.6.

The data saved in the microcontroller will be refreshed constantly and only the most recent value of each sensor is saved. For sending the data to the receiver, transmitter – receiver chips are used. These chips transmit data via radio and allow the development of an own communication protocol between both chips.

The final communication link is created between the receiver and the PC. The USART device is used to transmit the information. As referred before, the data is sent in ascending order based on the identifier number and in a constant manner as well. The PC interface is the final process of the telemetry system. An interface is created where the user gets the opportunity to benefit from the collected values. The system manages the received data in two different ways:

- The data is shown in real- time in the command- line window
- On the other hand, the data is saved in an excel table for future analysis. Each identifieer is saved in a different column. Thus, the data will be sorted out in a way where graphs could be drawn or the average measurement of a sensor is calculated.

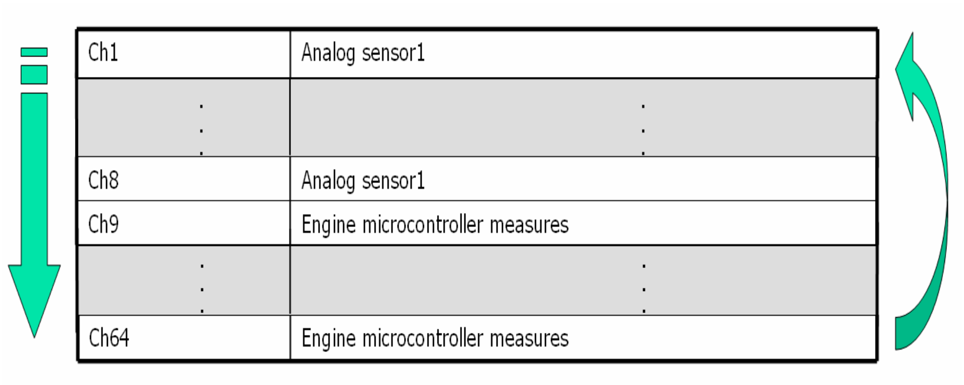

Ch1
Analog sensor1
Ch8
Analog sensor1
Ch9
Engine microcontroller measures
Ch64
Engine microcontroller measures

Figure 2.6: Transmitter- Receiver protocol[11]

2.3 Formula 1 Telemetry System case study

When we talk about Telemetry systems, we must refer the most sophisticated telemetry systems in the industry, the one used in Formula 1. Formula 1 is the most expensive of all the motor sports [12], therefore it is important that every component is constantly analyzed to get the best result possible.

Data acquisition in Formula 1 has suffered huge improvements in the last years, especially in 2002 when it was allowed for the first time to change parameters in the car from the pit. This technology is costly and became untenable for most of the teams, therefore it was changed again in 2003.

Magneti Marelli is a product supplier of the top teams of the Formula 1 [14]. This company provides products for all the motor sports and for the most important car companies. In Magnet Marelli, the telemetry system is developed in two different ways. It can be embedded in the engine control unit (ECU) or it can appear like an independent system that receives data from the ECU. The most common system in Formula 1 consists of embedding the control and the communication in the same box so they can save space and have more reliable system.

Since the speed of the car is very high and the distances to be covered are very long, different technologies have to be used to collect the information:

- Real time data acquisition: The most important parameters such as the ones necessary for safety issues are sent constantly to the engineers at the pit box. An aerial antenna located in the side pod nearest to the pit side is used to transmit the data. This system is powerful because it covers any circuits but the bandwidth isn't enough to send all the necessary data (over 150'000 measurements are made per second inside a formula 1 car during a race). Consequently, other communication medium has to be used to compensate the lack of bandwidth.

- Burst transmission. For all the data that is necessary during the race but not important in every second, burst transmission is used. Each time that the car passes close to the pit lane, a microwave burst is sent to the pits. In a race, this data burst contains between 2 and 5 Mbytes of information that are transmitted in two seconds. The collected information is used for calculations such as the optimal lap for refilling the car or for getting an average of parameters that are analyzed after a car evolution. Every parameter is also checked to make sure that the car won't suffer any main problems.

- The most important data are saved in a data logger to be examined after the race so that every part of the car is deeply analyzed. The usual way to download such information is through connecting a computer to a special plug.

- The data is not only used at the pit, but also sent to the headquarters of each team through satellite communications where the information is of major importance for the development of new car generations and for further tests.

The manufacturer that supplies the car's electronic components usually provides the graphical interface but some teams have their own software. ATLAS (Figure 2.7), which has been used by McLaren during the early nineties, is a good example.

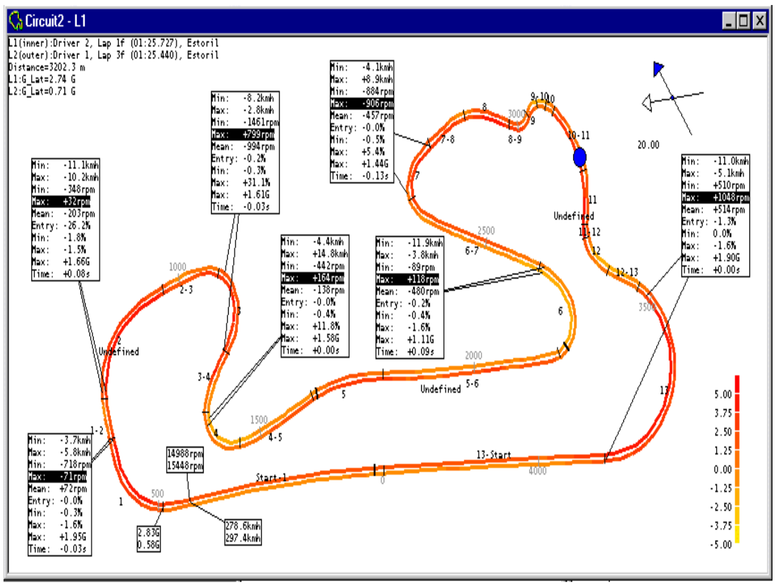


Figure 2.7: Atlas software developed by McLaren[13]

2.4 Communications

As refered above, the two most important parameters for a telemetry system are the maximum range of the communication medium and its bandwidth. These two parameters are determined by the communication link that is used.

We can divide the communication in two types: internal and external communications. Internal communication is the way the communication is made between the logical components of the system, while the external communications are the communication medium used to forward data to a remote location.

External communication can be divided in long- range and short- range. In a telemetry system, the long- range communications are used for upload the acquired data to a remote location, while short- range communications are mostly used to establish the connection with the car when the device is inside the vehicle.

An abstract architecture of our telemetry system, where all connections are labelled with a letter is depicted in figure 2.8:

A. Traditionally between the car unit and the car, we use a short- range communication, since the distance between both is short.

B. The communication between the car unit and the pit unit is classified as long- range communication, because the distance between them can be 100 meters or more.

C. In order to provide visualization to the driver, the component responsible for update the user interface must be provided with data. This type of communication is classified as internal communication, which we will address in chapter 4.

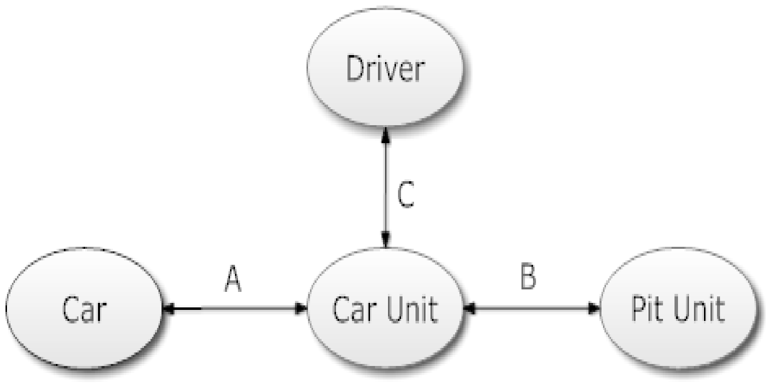


Figure 2.8: Communication abstract schema

2.4.1 CAN Bus - onboard communication bus

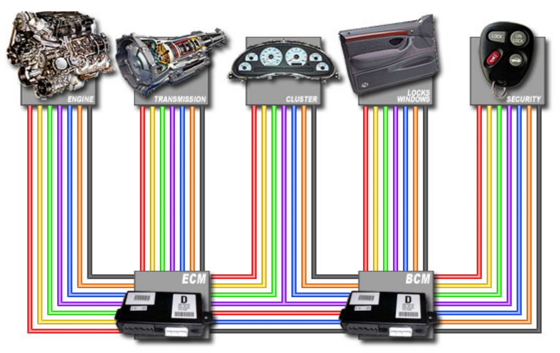
As referred above, the HammerShark vehicle is fitted with a CAN Bus [15]. This CAN Bus is responsible to collect all data produced by the sensors that are spread along the car.

Before Robert Bosch created the first CAN Bus, vehicles contained enormous amounts of wiring in order to interconnect all electronic components. Due to the amount of wiring (Figure 2.9(a)), an after-market installation requires the installer not only to understand how the integrated systems communicate with each other, but also requires numerous connections to be made throughout the vehicle.

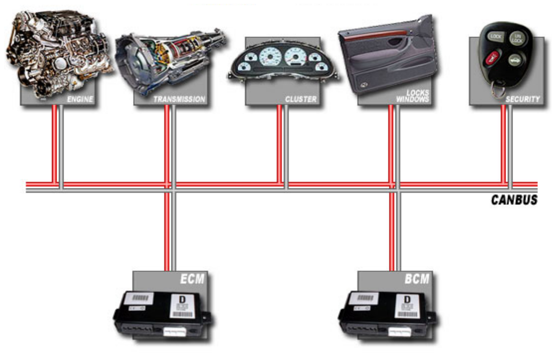
The goal was to make automobiles more reliable, safe and fuel- efficient while decreasing wiring harness weight and complexity (Figure 2.9(b)). Since its inception, the CAN protocol has gained widespread popularity in industrial automation and automotive/truck applications.

The CAN protocol was optimized for systems that needed to transmit and receive small amounts of information, reliably to any or all other nodes of the network. The protocol is message- based, where every message includes an 11 bits field (29 bits in extended mode). This field corresponds to an ID, which can be used as an address. Each node can define filters that allow the controller to accept or not the received message. This allows simultaneously the creation of node- to- node, multicast or broadcast messages.

On a CAN bus, faulty nodes will automatically drop off the bus not allowing to bring the network down by one node. This effectively guarantees that bandwidth is always available to transmit critical messages.



(a) Conventional wiring[16]



(b) CAN Bus network[16]

Figure 2.9: Automotive component wiring evolution

2.4.2 External Communication

We classify external communication as any communication that is established from the smartphone to another device. To achieve this type of communication we don't need additional hardware, since modern smartphones are fitted with WLan and Bluetooth modules. Several Android phones are even already fitted with a NFC (Near Field Communication) [54] chip.

Telemetry systems requires a communication link, mainly for providing data upload in order to be analyzed by team members. At the Shell Eco- Marathon the amount of data produced is low and therefore the teams can choose one wireless technology that fit their needs. However, at the formula 1 for example, over 150'000 measurements are made per second and every byte is important and worth money. In this case, the choice of the right protocol to communicate between the car and the pit lane is important.

We have several wireless communication solutions that we could use in order to establish a connection with a remote device, such as Wi-fi [17] [18], GSM [23] [24], UMTS [27] [25] and

LTE [31] [32]. These standards are operating system and platform independent, working on nearly every modern smartphones and computers. However, already exist a mechanism that are expressly used in Android devices, Android Beam [41].

2.4.3 Long- range communication

Long- range communication solutions are usually used for establishing communications to devices located faraway from our actual location. In our system, we classify as a long-range communication, the communication between the car unit and the pit unit (Figure 2.10).

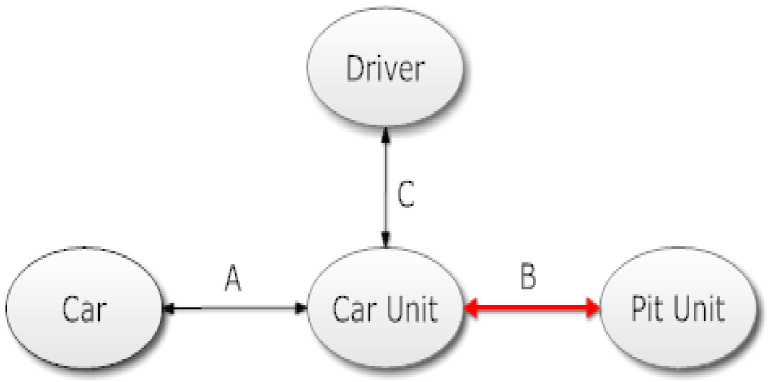


Figure 2.10: Long- range communication on the abstract scheme

Even with long- range communication solutions, since they are wireless, they have limited range. This limit depends on the transmission power, antenna type, location they're used in and the environment. There exist however, some solutions that can be used to communicate with remote devices that are several hundred meters away.

Smartphones are now fitted with a WLan module, allowing the device to connect to a WLan using the IEEE 802.11 standard [18]. Devices uses this technology for obtaining connectivity. Wi- fi [17] can be used in two different network topologies, ad-hoc and infrastructure.

With the network topology ad- hoc, all wireless devices can communicate directly with each other. The network is considered to be decentralized and there is no hierarchy between the devices. All devices in an ad- hoc network have equal status. All wireless devices are able to discover and communicate in peer-to-peer fashion between them. Unlike the ad-hoc topology, a infrastructure network needs an access point (AP). All wireless devices must be configured to use the same SSID. The AP manages the access to the network and to shared resources if any exist. With the infrastrucutre mode, the network can be scalable and has a centralized security management.

With the appearance of several wireless access technologies, ubiquotous computing has become a desire for mobile users. Wi- fi has a limited range and Wi- fi hotspots could not be near to our location. The alternative to provide mobile wide- are network access for mobile users are mobile networks (GSM, UMTS, LTE). Since we using mobile devices for our telemetry system, the use of such a technology for data transaction is a possibility.

GSM (Global System for Mobile Communications) [23] [24] is a standard set developed by the European Telecommunications Standards Institute (ETSI) with the aim of replacing the first generation analog cellular networks. GSM is considered to be the second generation digital cellular network. Unlike in first generation celllular networks, data services are an integral part of a GSM network and are supported together with ordinary voice services. Therefore, he can be used for Internet access. In fact GPRS (General Packet Radio Service), which is a packet- switched data service standardized for GSM was the mos powerful wireless Internet access technology by the turn of the millennium. GPRS users has the advantages that they are always on- line, can dynamically allocate bandwidth also in an asymmetric fashion on up- and downlink and pay per transmitted/received data volume. The benefits for the GSM operators offering GPRS are highly efficient and cost- effective use radio spectrum and network resources.[22]

With the ubiquotous computing, the users become the need of accessing all type of data content from the smartphone. This means that the mobile phone market needed a technology that provides higher data rates. In order to address the new needs of the mobile phone market, the UMTS standard [27] [25] was developed. UMTS is classified as the third- generation cellular network and provides higher data rates along with real time voice calls. UMTS provides data rates up to 2 Mb/s in indoor or small- cell outdoor environments, and wide- area coverage of up to 384 kb/s. High Speed Downlink Packet Access (HSDPA) is a communication protocol, which allows cellular networks based on UMTS achieve higher data transfer speeds and capacity.

After UMTS, the 3rd Generation Partnership Project (3GPP) introduces the fourth generation cellular network. Named LTE (Long Term Evolution) and based on GSM/EDGE and UMTS/HSPA network technologies. LTE increased the capacity and speed of wireless data networks by using new modulation techniques. With LTE is is possible to offer higher data rates than the ones provides by UMTS to mobile users.

The LTE [31] [32] specification provides downlink peak rates of 100 Mbit/s, uplink peak rates of 50 Mbit/s and QoS provisions permitting a transfer latency of less than 5 ms in the radio access network. The IP- based network (Envolved Packet Core) architecture was implemented with the aim to replace the GPRS Core Network, supporting seamless handovers for both voice and data to cell towers with older network technology such as GSM and UMTS.

	WLan	GSM	UMTS	LTE
Frequency	2.4+ Ghz and 5.15 Ghz	900 Mhz and 1800 Mhz	1.92 Ghz (Upstream) until 2.17 Ghz (Downstream)	In Europe 800, 900 Mhz and 1.8, 1.9/2.1, 2.5 Ghz
Bandwidth	54 Mbit/s	9.6 Kbit/s	With HSPA: Downstream: 7.2 Mbit/s Upstream:1.45 Mbit/s	Downstream:100 Mbit/s Upstream:50 Mbit/s
Range	30-100 m	35 Km free view 100 m in the city	6 km	2 km in dense area

Table 2.2: Comparison between long- range wireless solutions

In order to make the right choice for support data transmission in a telemetry system, Table 2.2 shows a comparison between all technologies described above.

2.4.4 Transport Protocol

All of these wireless solutions referred above, only implement the physical layer of a network. In order to provide connectivity between the devices that comprises our telemetry system, we need to choose a transport protocol in order to transfer data between them. There are two protocols that can be used to implement the transport layer, TCP and UDP.

Transfer Control Protocol (TCP) [47] is caracterized to be a reliable protocol, which includes techniques that guarantees that all packets are delivered to the destination. TCP is connection- oriented, this means that before an application process can start sending data to another, the two process must first "handshake". They send some preliminary segments to each other to establish the parameters of the data transfer. Only after the connection was succesfully established, the protocol starts to transmit the packets. If the packet was received by the destination, an acknowledge is received by the sender. In order to know if each packet was acknowledged by the destination, the sender keeps a timer from when the packet was sent and retransmits the packet if the time runs out. This technique is called positive acknowledgement with retransmission.

When using TCP, data can be split into chunks. This happens if the data to send is greater than the maximum transmission unit (MTU). TCP pairs each chunk with a TCP header, forming a TCP segment. The segments are then encapsulated within Internet Protocol datagrams at the network layer. The Internet Protocol datagrams are then sent to the destination.

The fields of a TCP segment are depicted in figure 2.11. The first 16 bits identify the source port, and the second 16 bits identify the destination port. source and destination port numbers are used to multiplexing/demultiplexing data from/to upper- layer applications. The sequence number and the acknowledgment number are two reliability mechanisms.

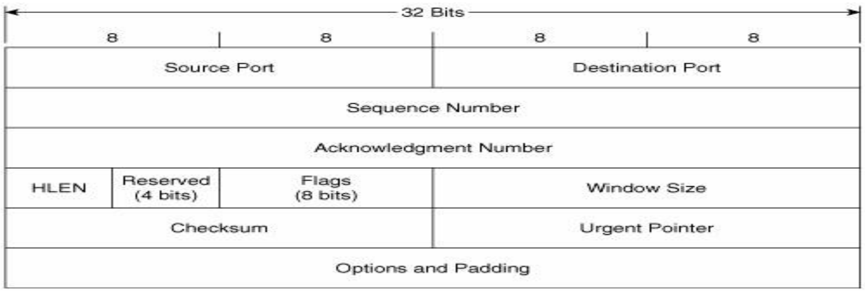


Figure 2.11: TCP Segment[45]

Contrary to TCP, UDP [47] is a connectionless protocol. In UDP there is no handshaking with the UDP entity running on the destination end system. The sender adds header fields to the data, creating an UDP segment. The resulting UDP segment is then encapsulated into a datagram , which are then sent to the destination. Since there are no mechanisms that guarantees the delivery of the datagram, the sender can't know if the datagram was delivered or not.

If we compare the UDP segment's structure depicted on figure 2.12 to the one of TCP that is depicted in figure 2.11, we can observer that UDP doesn't have the reliability and control mechanisms of TCP.

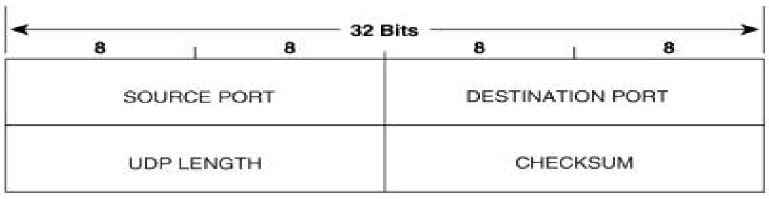


Figure 2.12: [45]

In this context, when the main concern is data throughput, UDP may be the best choice. Although there is no data integrity control. On the other side, TCP is clearly the protocol of choice, because it provides data integrity, controllability and reliability control.

2.4.5 Short- range communication

Communications are classified as short- range if the device, with which we want to establish a connection is in the range of less than 10 meters. At our telemetry system, the communication between the HammerShark vehicle and the car unit (Figure 2.13), can be classified as short- range communication. Since the distance between the car and the car unit is less than 10 meters. When our target is so close, we have the ability to use wireless solutions with fewer range. IrDA [51] [50], Bluetooth [43] and NFC [54] are wireless solutions that are classified as short- range communication solutions and that could be used to provide connectivity between the car and the car unit at our telemetry system.

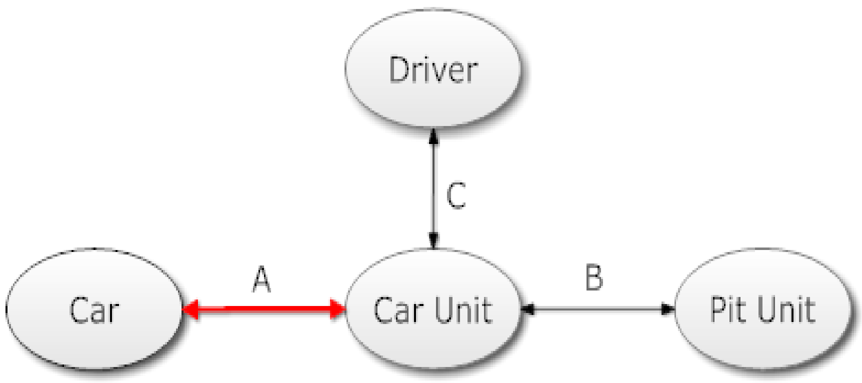


Figure 2.13: Short -range communication abstract scheme

Infrared Data Association (IrDA) [51] [50] specifies a set of protocols for infrared data communications. IrDA conform devices use an inexpensive and widely adopted short- range wireless communication technology that allows devices to communicate with each other. For establishing a connection between both devices, devices are classified as primary and secondary. The primary device is responsible for selecting a device, establishing a connection, and maintaining the link. The secondary device only responds to demands of the primary device. When the primary device wants to establish a connection, he initiates a process known as "discovery", searching for available devices that are near. From those devices that respond, the primary selects a device and attempts to connect to it. During connection establishment, the two devices negotiate to understand each other's capabilities. In this way a connection can be optimized despite the differences between two different devices. Once they have negotiated, they will adopt the highest common transmission speed, and attempt to communicate.

At this point, applications on either side of the connection can transfer data. Infrared data communications operate in half- duplex mode, because while transmitting, a device's receiver is blinded by the light of the transmitter and therefore full duplex communication is not practicable. Since infrared uses light waves for communication and light waves can't pass through a solid objects, both devices must be on line of sight.

The drawback of having to fix both devices in line of sight led us to look for alternatives for short –range communications. In this aspect Bluetooth [43] might be a better solution, it has more range than Infrared (10m against 1m) and the devices don't need to be in line of sight. Bluetooth is a global standard for wireless connectivity, based on a low- cost, short-range radio. Bluetooth technology facilitates the replacement of the cables normally used to connect one device to another, with one universal short- range radio link.Two Bluetooth devices can talk to each other when they come within a range of 10 meters. Due to their dependence on a radio link, as opposed to alternate technology such as an infrared connection, Bluetooth devices do not require a line- of- sight connection in order to communicate.

The Bluetooth technology supports both point- to- point and point- to- multipoint connections. When two Bluetooth devices are connected, we have a piconet [53]. A piconet starts with two connected devices and supports up to eight devices that are connected in ad-hoc fashion. In a piconet there will be a device that is considered as a master, while the remaining devices that comprises a piconet are considered slaves for the duration of the connection. When two piconets have overlapping coverage areas, then we have a scatternet [34]. Two Bluetooth devices can be part of two piconets. Slaves in one piconet can participate in another piconet as either a master or slave. In a scatternet, the two (or more) piconets are not synchronized in either time or frequency.

Even if Infrared and Bluetooth are platform independent, there is also a technology with the ability to exchange content using only Android devices – Android Beam. Android Beam is an NFC based technology with which the user can share data by simply tapping the two devices together within the reach of NFC range. The disadvantage of this technology, is theinfrared need of two phones fitted with Android OS and with a NFC module. However, not every device will be available with an NFC module. This technology is also exclusive for Android operating systems in the version 4.0 of the OS (Ice Cream Sandwich).

NFC [54] is a Radio-frequency identification (RFID) [33] based technology for contactless short- range communication, operating in the 13.56 MHz frequency band. The communication between devices is achieved by using magnetic field induction. NFC supports transfer rates of 106, 212 and 424 kbps. Even if NFC was designed for communications up to a distance of 20 cm, typically it is used within less than 10 cm. It can operate in two different modes: active and passive. In active mode, the device generates its own RF field, while a device in passive mode has to use inductive coupling to transmit data. Contrary to active mode, in passive mode no internal power source is required.

Compared to other short- range communication technologies, which have been integrated into mobile phones, NFC simplifies the way consumer devices interact with another and obtains faster connections. While IrDA, the oldest technology, is the fact that a direct line of sight is required, which reacts sensitively to external influences such as light and reflecting objects. The significant advantage over Bluetooth is the shorter set- up time. Instead of performing manual configurations to identify the other's phone, the connection between two NFC devices is established at once (<0.1s). Table 2.3 points out these different capabilities of NFC, Bluetooth and IrDA. All these technologies are point- to- point protocols. Bluetooth also supports point- to- multipoint communications. With less than 10 cm, NFC has the shortest range. This provides a degree of security, turning NFC suitable for crowded areas. The data transfer rate of NFC (424 kbps) is slower than Bluetooth (721 kbps), but faster than IrdA (115 kbps).

	IrDA	Bluetooth	NFC
Network Type	Point-to-point	Point-to-multipoint	Point-to-point
Range	1m	10m	<0.1m
Speed	115kbps	721kbps	424 kbps
Setup time	0.5s	6s	<0.1s
Modes	Active- active	Active-active	Active-active, active-passive
Costs	Low	Moderate	Low

Table 2.3: Comparison between short- range technologies[54]

Chapter 3

DroidShark

Implementing a Telemetry System for a vehicle that competes in a race event is expensive and time consuming. Many teams build custom hardware in order to execute this task. Our approach will address this problem, proposing a new Telemetry system using mobile devices running on Android OS. The proposed system will provide more interaction between the team members and the driver, without the need to use custom hardware to interact with the vehicle. Data acquisition, processing and relay are tasks that will be executed by the mobile device.

3.1 The scenario

DroidShark was conceived for one specific scenario and to work as a telemetry system in order to support the team members and the driver during a competition. The vehicle is equipped with a CAN bus, which collects all information produced by the internal sensors. All information can be accessed through the Bluetooth interface and is sent encapsulated in well-formatted data streams.

DroidShark consists of two mobile devices, a smartphone and a tablet. The smartphone (car unit) is located at the driver's steering wheel and the tablet (pit unit) is located at the pit lane among with the remaining team members (Figure 3.1). The smartphone establishes a connection with the vehicle and starts to retrieve data. All data are processed in live time, giving visual feedback to the driver, storing all information on files at the phone's external memory and transmit all gathered data to the pit unit. The pit unit gives visual feedback to all team members, based on the information retrieved from the car unit and provides also data logging of the retrieved data. In our system, we will store the same information on both devices - smartphone and tablet. These log files will be then used for race analysis after each race. The whole race can be simulated at the pit unit, because it is the only device that allows this feature. The tablet was choosen, because it is fitted with a greater display than the smartphone, turning it the optimal solution for viewing all available data in only one screen.

At each instant the team members can send information to the driver, by typing a message on the pit unit and sending it to the car unit. The car unit, which will be always listening for incoming messages, will then process the incoming message, showing it to the driver.

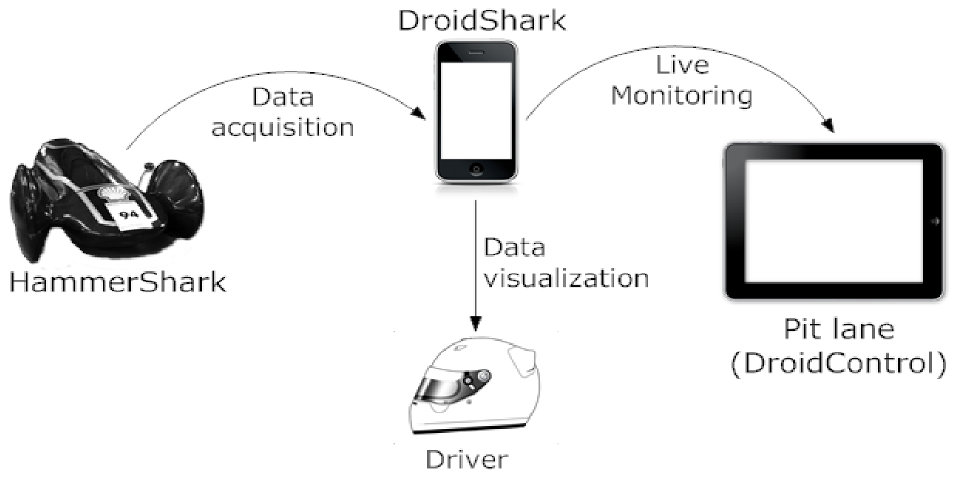


Figure 3.1: DroidShark Overview

3.2 The architecture

As we referred above, the actual instances of DroidShark run on a smartphone and in one tablet. Therefore the DroidShark architecture can be split in three main modules:

- HammerShark, which is fitted with a CAN bus and allows accessing all related information via a Bluetooth module.
- The car unit is responsible for the data acquisition from the vehicle, but also used as a driver display.
- The pit unit, offers live monitoring of the vehicle data for the team members at the pit lane.

HammerShark is fitted with a CAN bus and can be compared to a black box, where all data can be accessed via a Bluetooth interface. The unit responsible to interact with HammerShark is the car unit. The car unit is responsible for processing the information and provide the driver with useful data in order to accomplish the race goals. The car unit is also used as a gateway to the pit unit, that abstracts the car unit as data source - as in car unit towards the CAN bus.

DroidShark focus mainly on the following aspects:

- Gather information from the HammerShark vehicle
- Provide an user interface that presents useful information for the driving (e.g. speed, rpm, distance covered, time elapsed)
- Logging mechanism for the acquired information
- Serve as gateway for remote devices
- Provide support for the team members to send text messages to the driver

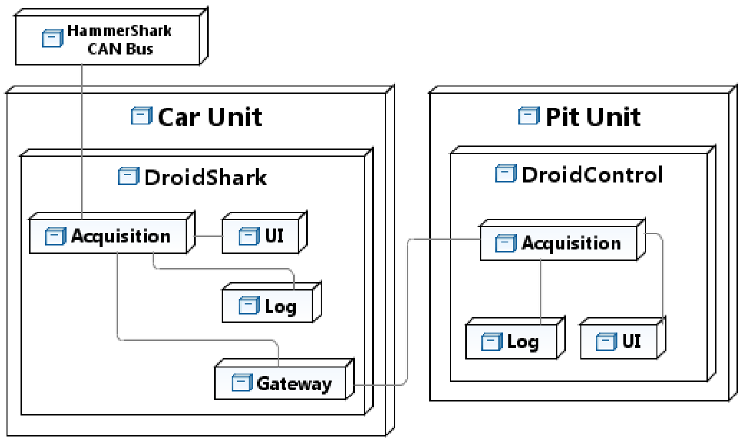


Figure 3.2: DroidShark Component Diagram

To address these requirements we propose a new architecture that we called DroidShark. DroidShark is based on both mobile devices (smartphone and tablet) and in order to take the advantage of the IPC ability of the Android operating system, we propose a modular architecture (Figure 3.2) that is composed by:

- Acquisition module
- Logging module
- UI module
- Gateway module

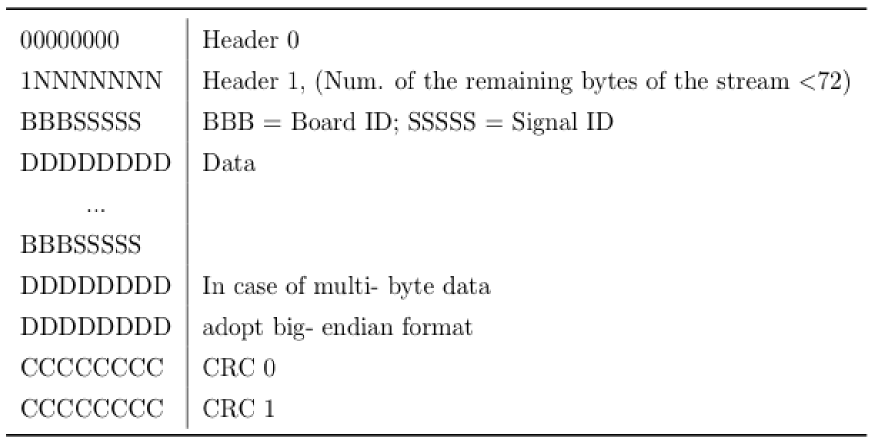
All modules were implemented as background services that are started together with the application. Background services have the ability to run in background without user interaction, which we can compare with Windows services or Unix deamons.

The architecture followed is the same on the car unit and on the pit unit, which besides running on a tablet is similar technologically speaking. Therefore all these modules enumerated above can be found on both devices, almost executing the same task.

3.2.1 Acquisition module

The acquisition module is available at both devices. However, the data sources are different. On the car unit, the acquisition module is responsible to acquire data directly from the HammerShark vehicle, through its Bluetooth interface. HammerShark sends all data encapsulated in streams (Figure 3.3), which must be processed by the acquisition module in order to obtain the value produced by each sensor of the vehicle. After the received stream has been processed, the module must disseminate all obtained values to the remaining modules that comprises the unit.

Despite acquiring and processing the information from the HammerShark vehicle, the car unit is also responsible to serve as gateway to the pit unit. Therefore the difference between the acquisition module that can be found at the car unit and the one that can be found at the pit unit is the data source. While the car unit receives data directly from the vehicle, the pit unit depends on the data sent by the car unit. Despite the different data sources, the acquisition module on both devices treats and sees the data similarly.



00000000	Header 0
1NNNNNNN	Header 1, (Num. of the remaining bytes of the stream <72)
BBBSSSSS	BBB = Board ID; SSSSS = Signal ID
DDDDDDDD	Data
...	
BBBSSSSS	
DDDDDDDD	In case of multi- byte data
DDDDDDDD	adopt big- endian format
CCCCCCCC	CRC 0
CCCCCCCC	CRC 1

Figure 3.3: Aspect of the streams sent by the HammerShark vehicle

3.2.2 Logging module

The logging module, similar to the acquisition module described above, can be found either on the car unit and on the pit unit. Since data logging is important and the produced data can be used to avoid failures in future competitions, this module performs the same task on both devices, at the same manner. At the telemetry system, the logging module is responsible to store all data received from the acquisition module into log files. These log files are used for data analysis after the race, which can lead to improvements of the performance in future challenges.

3.2.3 Gateway module

As referred above, the DroidShark comprises two mobile devices. One mobile device - the car unit- communicates directly with the HammerShark vehicle. However, the pit unit must be provided with the acquired data. In order to provide the pit unit with data, the gateway module has been implemented. The gateway module is responsible for providing all data received from the acquisition module to the pit unit, using a long- range communication solution.

Despite providing the pit unit with the acquired data, the gateway module is also responsible for receiving incoming messages sent by the pit unit. These messages are text messages designed to be delivered to the driver. Therefore, this module is also responsible to forward all received messages, to the module responsible to provide data visualization to the driver.

3.2.4 UI module

In a race like the Shell Eco- Marathon, each team has race goals to accomplish. Therefore, it is important that the driver has the knowledge of how he is doing during the race. In order to provide data visualization to the driver, the UI module has been implemented.

The UI module is responsible for providing data visualization to the driver, of all the relevant data received from the acquisition module. Some of this displayed data is actually pre-processed or derived data. However, the gateway module is also a data source of the UI module, since it is the gateway module that receives messages sent by the pit unit ot the driver.

Chapter 4

DroidShark Implementation

In this section we will present the overall implementation of DroidShark, discussing some implementation options – mainly at IPC level, supported on trials and experiences performed on a simulated environment.

DroidShark consists of two Android applications, one smartphone application called DroidShark, which will be running on the car unit. The other one is a tablet application called DroidControl, which will be installed on the pit unit. Since both applications are Android based, the architecture will be similar. However, they differ at IPC level implementation.

4.1 Android

From the beginning, it was established that both applications would be supported on the Android OS. Android OS is an open source mobile operating system and is currently supported by the Open Handset Alliance [57]. Android has several features that we found interesting and were very relevant in the design and implementation of DroidShark mobile applications:

- Support multitasking and background services: This feature allows maintaining several threads executing simultaneously and allow the implementation of a modular architecture for an application.
- Abstracts hardware and network resources: Android abstracts through API's the access to hardware devices such as GPS, accelerometers and digital compass. Other services such as Wi- fi, Bluetooth and Telephony, also provides specific APIs access.

- Data sharing and inter process mechanisms: The high level concepts of intents and AIDL allow communication between third- party applications. This is especially useful to decouple the data client from the data provide details at the Android OS level.

- Open source framework: Android OS possesses a specific SDK.

4.2 DroidShark

The car unit establishes a connection with the car and acquire all data produced by the vehicle during the race (Figure 4.1). As referred above, the car unit is an application running on an Android operating system based smartphone. The application is named DroidShark.

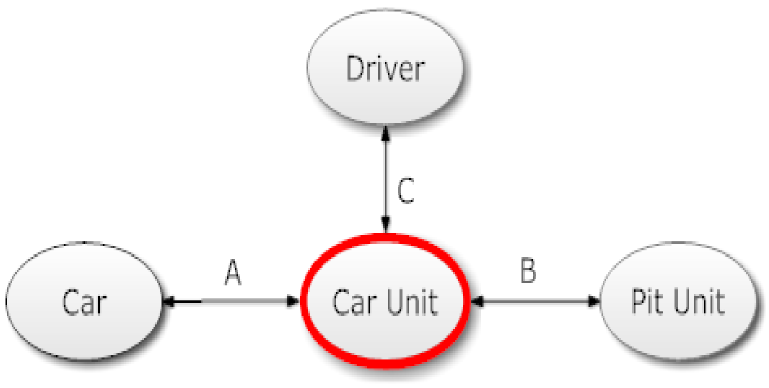


Figure 4.1: DroidShark on the abstract scheme

DroidShark follows the Android computational model [56], therefore the application is structured on several *activities*. Background services are also implemented, in order to provide a modular architecture for the telemetry system.

The main DroidShark *activities* (Figure 4.2) are:

- **Property Activity:** This activity is the first screen that appear to the user when he starts the application. On this activity he must provide some configurations for the proper work of the application. All provided configuration is validated before the application takes the next step.

- **DroidShark Activity:** The DroidShark activity is the central activity of the application. This activity is responsible for starting all modules that comprises the application. The DroidShark activity also creates the three screens for the user, which he can switch by using a button on his steering wheel.

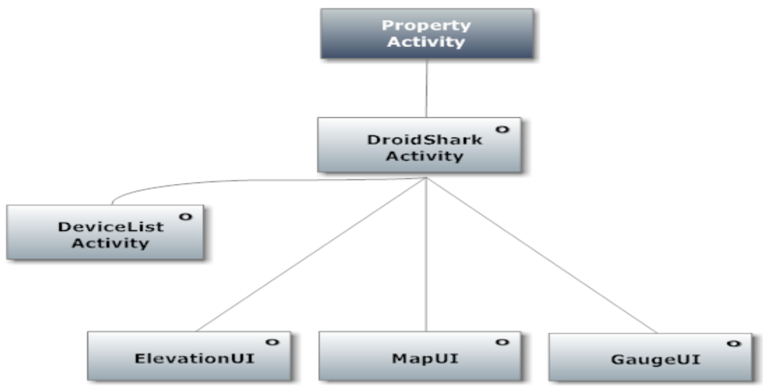


Figure 4.2: DroidShark activities

- **DeviceList Activity:** This activity is responsible for searching available Bluetooth devices in the near range area, which are available for pairing and for establishing a connection.

4.2.1 DroidShark User Interface

DroidShark user interface is responsible for providing the driver with the acquired information, as depicted in Figure 4.3 on our abstract scheme of the telemetry system. The user interface is the contact point between the driver/user of the application and the telemetry system. Despite providing the driver with the acquired data during the race, the user interface offers also the ability to configure settings, namely name of the log file, ip address of the pit unit, port for incoming transmissions and selecting the Bluetooth device to connect. The design of the user interface is based on requirements identified in previous editions of the Shell Eco- Marathon. The evelvation profile of the track and the aerial persepective of the track with the actual position of the vehicle must be part of the user interface.

When the user starts the application at the smartphone, a configuration *dialog* appears (Figure 4.4). The configuration *dialog* is divided into several areas, namely the forward options, log options and a track options. At the forward options the user need to insert the *ip* address of the remote device and the port number where the car unit is listening for incoming messages from the pit unit. The user can also define the port where the car unit is listening for incoming connections.

At the log options the user must specify the name of the log files that are created later, when the log module is started. By specifying the name of the file, the user can maintain the control of the files that are used for data logging. Despite the log options, the configuration *dialog* also includes a track options.

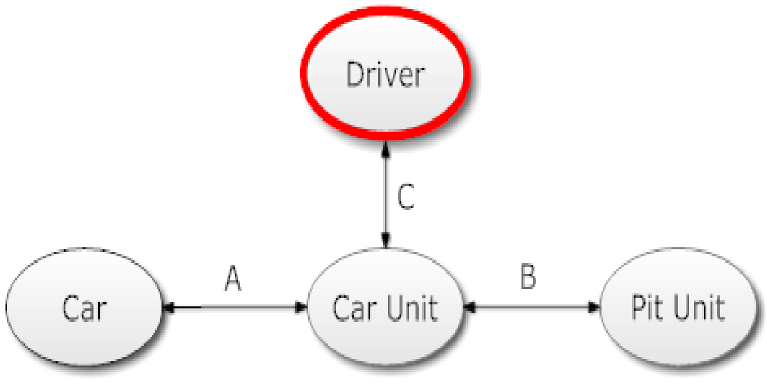
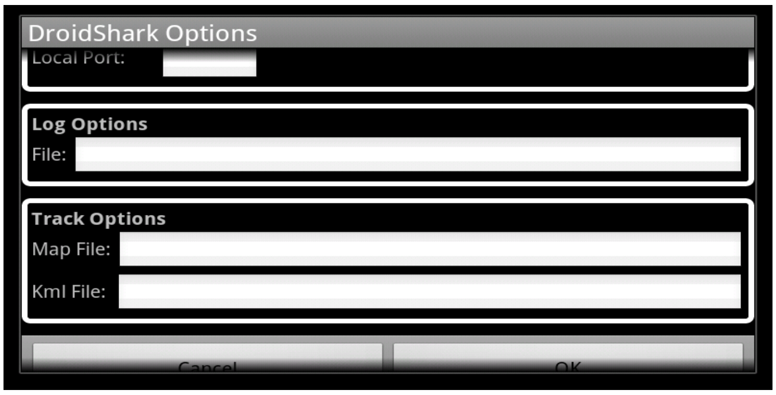


Figure 4.3: DroidShark UI at the abstract scheme

We are using offline maps for illustrating the track. For the use of offline maps, the user must specify the path of the map file at the track options area. A KML file [61] with the description of the track must also be specified at the track options.


DroidShark Options

Local Port:

Log Options

File:

Track Options

Map File:

Kml File:

Figure 4.4: Options dialog of the DroidShark application

After the user has set all configuration fields with valid information, the main driver UI is provided to the user. Since it is a big amount of information, the best solution is to divide the information through different screens. Therefore the main driver UI has three different screens that can be accessed by the driver, where a button on his steering wheel is used to switching between them. The three screens aren't activities, since the overhead by switching between each screen were too big. We implemented all screens using a view switcher. Using this solution all three screens are loaded at once, but resources like images or maps of a given screen are only loaded if this screen is currently viewed by the driver. For example, if the driver is viewing the screen one, screen two and three are hidden and therefore all images from screen two and three are freed. The same procedure occurs by updating the screen, when a new stream is processed and sent to the UI module that is responsible for updating the user interface, only the screen that is presented to the driver is updated. All other screens that are hidden aren't updated with the new data.

The information that is presented to the driver is divided across all available screens, but there is some information that is important and therefore must be available on all three screens. Information like instantaneous speed, current lap, time elapsed and distance covered can be found on all screens. With the presence of these informations on every screen, we prevent the driver to switch the screen every time he need to consult one of these parameters.

Main screen

This is the first screen (Figure 4.5) that is presented to the driver after the configuration *dialog*. This screen is composed by gauges and labels, where the instantaneous speed, battery voltage and the rpm are gauges. A gauge is a combination of two images, a background image with a scale and an image of a needle on top. In order to move the needle to the pretended value, RotateAnimations [62] are used. Beside instantaneous speed, battery voltage and rpm, this screen also provides information of the lap number, time elapsed since the beginning of the challenge, distance covered and mean speed.

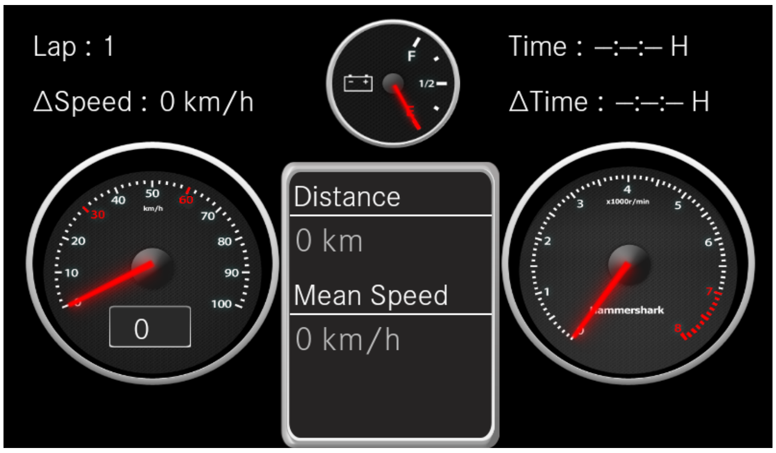


Figure 4.5: Main screen of the DroidShark application

Map screen

The map screen (Figure 4.6) was implemented in order to inform the driver in which part of the track he is at a given moment. The map screen presents an aerial perspective of the track implemented using offline maps.

Offline maps become an alternative to online maps like google maps, based on the lack of network connection. An offline map doesn't need a persistent network connection, and we could choose one area of a map to download and use as long as we want. In our case we only need to download the part of the map, where the race track is. Based on these characteristics, we researched for solutions and found OpenStreetMap [60].

OpenStreetMap is an editable map of the whole world, which is released with an open content license. This license allows free access to map images and all underlying map data. At the web page it's possible to choose the area that we want and download it in different formats. After downloading, the map is acting like a figure, however it has enough information for extracting gps coordinates. Even if we have the map, the Android API of a map view [63] must provide the ability to work with offline maps, but it doesn't. After some research we found an open source project with the ability to create overlays on top of the offline map – Mapsforge [58].

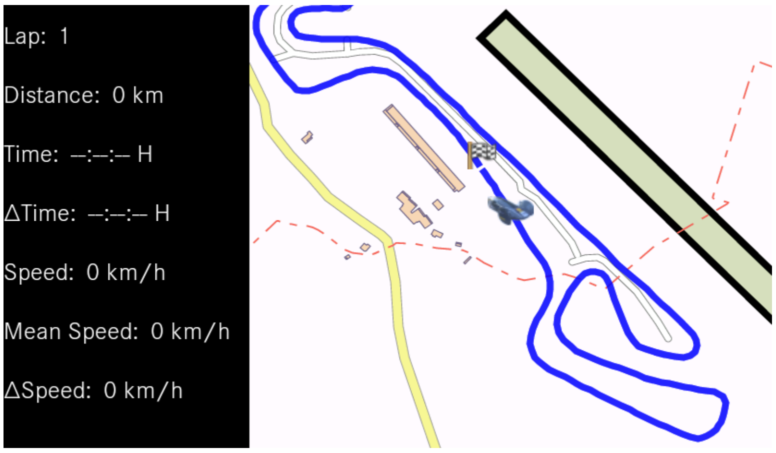
Mapsforge is a project that provides free and open software for OpenStreetMap based applications. The API has the advantage to be very similar to the Google Maps API [64], facilitating its use. With Mapsforge we created some overlays on top of the map, namely the track, car's position and the finish line. For drawing the track and insert a push pin for representing the finish line, we must have a list of coordinates that the user must provide. Therefore we use Google Earth [65] to draw the track, and to produce a KML file that we go to use to draw the track on top of the map. The file is chosen by the user in the configuration dialog at the start of the application. Since KML files are structured as xml files, we use the SAXParser [67] to parse the file as soon as the UI is loaded. Apart from the track and the finish line, the car's position is also represented at the map. The coordinates of the car are retrieved among with the other sensor data from HammerShark. The GPS coordinates are then sent to the UI module by the acquisition module, and if the location of the vehicle has changed, its location is updated on the map.The position is updated every second.

Apart from the map, this screen also includes some information that we consider relevant to the driver, such as instantaneous speed, mean speed, current lap number, distance covered and time elapsed since the beginning of the race.

Elevation screen

The elevation profile of the track is important, when we're trying to save fuel. When the driver knows that the next meters are a decline, he can stop the motor und save fuel in those meters. The Elevation screen (Figure 4.7) aims to provide the driver with such information. The elevation profile is represented in a XY graph that is draw with the use of the AndroidPlot [59] framework.

AndroidPlot is a pure Java API for creating dynamic and static charts within the Android application. It's designed from the ground up exclusively for the Android platform. The



Lap: 1

Distance: 0 km

Time: –:–:– H

ΔTime: –:–:– H

Speed: 0 km/h

Mean Speed: 0 km/h

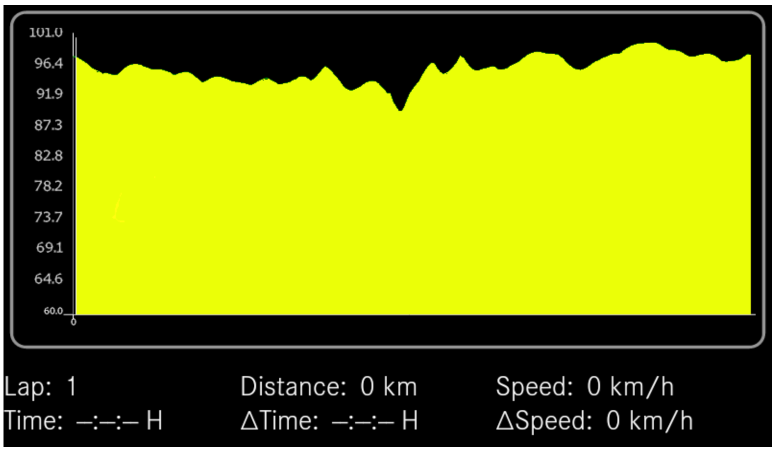
ΔSpeed: 0 km/h

Figure 4.6: Second screen of the DroidShark application

elevation profile is a XY graph, where the y- axis represents the altitude and the x- axis the distance from the finish line. Since we want to give the driver the sensation of continuity, we're using dynamic charts to provide this sensation.

For representing the car's position on the graph, we use a white marker. This marker is updated when the car's location is changed, by remove the marker from the current position and draw a new one at the new position. When the new position is beyond the middle point of the graph, the background is shift to left, until the marker is behind the middle point. We're ensuring with this strategy that the marker is always near the middle of the graph, turning it easier to the driver to identify the position of the marker.

Beside the graph on top of the screen, other parameters are provided to the driver, namely Current lap number, distance covered, instantaneous speed and time elapsed.



Lap: 1 Distance: 0 km Speed: 0 km/h

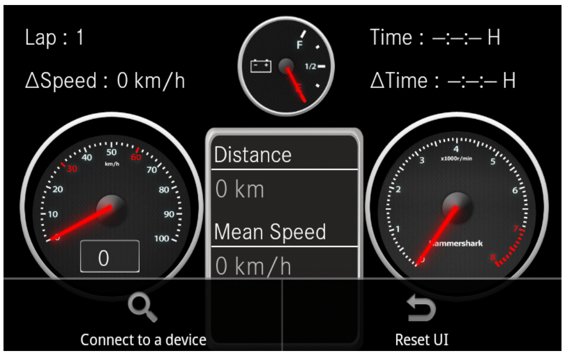
Time: –:–:– H ΔTime: –:–:– H ΔSpeed: 0 km/h

Figure 4.7: Third screen of the DroidShark application

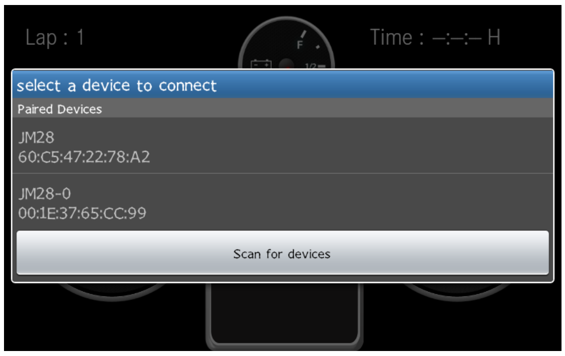
Establishing a connection with HammerShark

The connection with the HammerShark vehicle is established using Bluetooth. The use of this protocol requires that the user must search manually for available devices. For start a connection with HammerShark, the user must click on the menu button and select the option "Connect to a device" 4.8(a). After the click, a new activity window opens.

The new window is divided in two areas 4.8(b). At the top we can find a list of earlier paired devices, but if our device is not in that list we need to perform a new search, by clicking on the button "scan for devices". The results of the search are listed at the second area, which is below the first one. To choose a device, the user must click on the intended device. After the click the window is closed and the user returns to the previous screen. The mac address of the selected device is sent to the DroidShark activity, which is then provided to the acquisition module for establishing the connection and start data acquisition.



(a) DroidShark menu



JM28
60:C5:47:22:78:A2

JM28-0
00:1E:37:65:CC:99

(b) DeviceListActivity for search

Figure 4.8: Establishing a connection with HammerShark

4.3 DroidControl

Pit unit is the entity responsible for providing the visualization of the car telemetry data to the remaining team members at the pit lane (Figure 4.9), in form of a tablet. The application that is running on the tablet is called DroidControl.

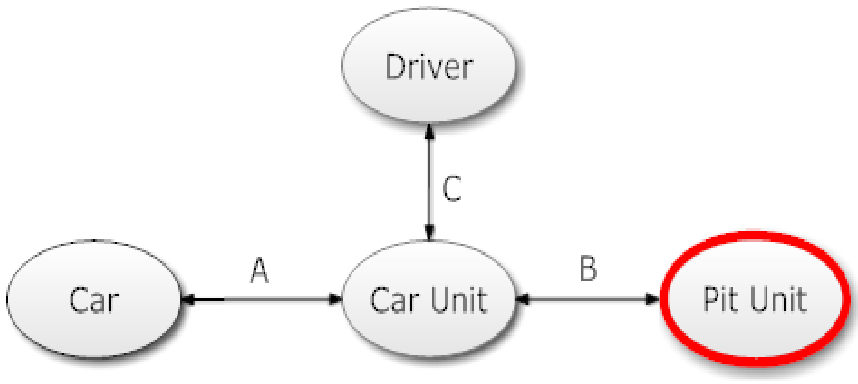


Figure 4.9: DroidControl in our abstract scheme

DroidControl is an application based on Android operating system. Despite the screen size that must be adjusted, the programming model is the same as for a smartphone application. Following the Android computacional model [56], the application is structured in several *activities* that support the user interface, and also services, which handle the tasks of data logging, remote communications and provide information for the user interface.

The main DroidControl *activities* (Figure 4.10) are:

- **Main Activity:** It's the activity that is shown when the user starts the application. In this activity, the user chooses which task he wants to execute, live acquisition or offline analysis of the previous acquired data.
- **DroidControlActivity:** Is responsible to start all background services and stop them when the user closes the application.
- **PropertiesActivity:** Used by the user to set some properties. The name of the log file, ip address of the remote device, port where the application is listening for incoming connections and the path of the offline map are some options that the user needs to set. This activity is shown as a dialog.

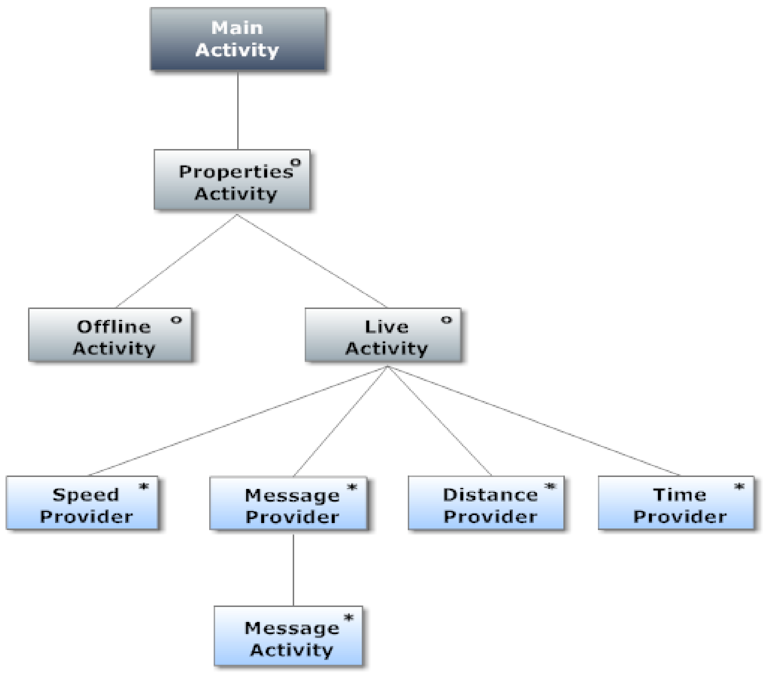


Figure 4.10: DroidControl activities

DroidControl offers an alternative option to view the telemetry data of the car, namely home screen widgets. Therefore we created some home screen widgets:

- **DistanceProvider:** This widget is responsible for providing information about the distance covered by the HammerShark vehicle during a competition.
- **MessageProvider:** We support the sending of text messages to the driver during a competition. For that purpose we developed this widget, which allow the team members to open a message activity by taping on the widget.
- **SpeedProvider:** SpeedProvider is responsible for providing the visualization of the instantaneous speed.
- **TimeProvider:** This widget is responsible for providing the information of the time elapsed since the start of the challenge.

4.3.1 DroidControl User Interface

A typical DroidControl user starts the application and the main screen (Figure 4.11) appears. The user has two options, analysing previous acquired data and start live monitor telemetry data during a challenge.

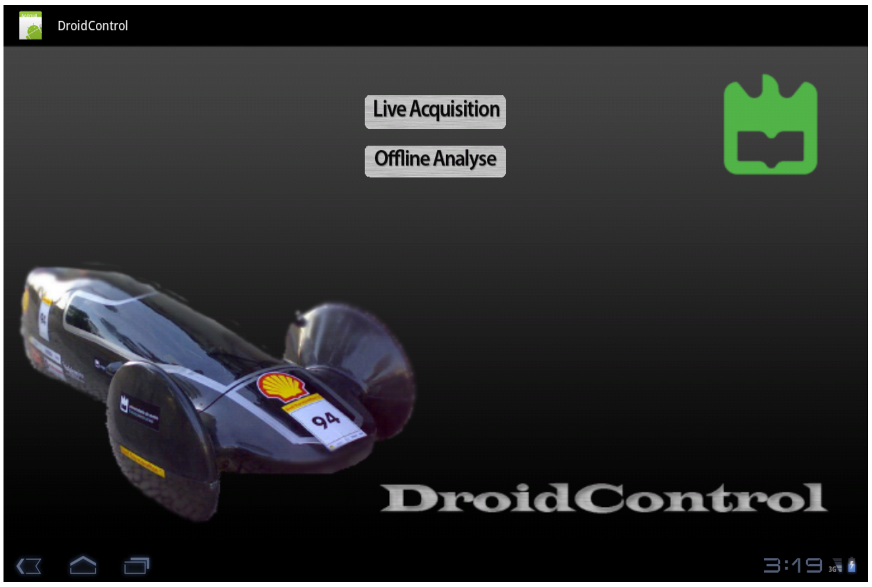


Figure 4.11: Main screen of the pit unit

Despite the user interface that the application provides, DroidControl also offers the ability to the team members to monitor the vehicle by using home screen widgets (Figure 4.12).

Home screen widgets offer one more way of presenting frequently changing information on the home screen of Android. From a high- level perspective, home screen widgets are disconnected views that are displayed on the home screen. A widget look and feel is defined through a layout XML file. For a widget, in addition to the layout of the view, the developer need to define how much space the view of the widget will need on the home screen. A widget definition also includes a couple of Java classes that are responsible for initializing the view and updating it frequently. These Java classes are responsible for managing the life cycle of the widget on the home screen, responding when the widget is dragged onto the home page and when dragging the widget to the trashcan.

Only the most relevant information like distance covered, speed and time elapsed are available in home screen widgets. Beside providing visualization of the acquired information, we developed a widget for sending text messages to the driver. When we developed the widget for sending text messages, we faced a problem by using home widgets. Home widgets aren't a regular activity, therefore it isn't capable to handle user input. This problem is caused, because a home screen widget isn't a regular *activity*, but an *AppWidgetProvider*. This means, we couldn't use a simple edit text widget for writing the text message that would be sent to the driver. In order to get around this limitation, we created an Activity in form of a dialog (Figure 4.13) that is started by clicking on the message widget. When the user clicks on the

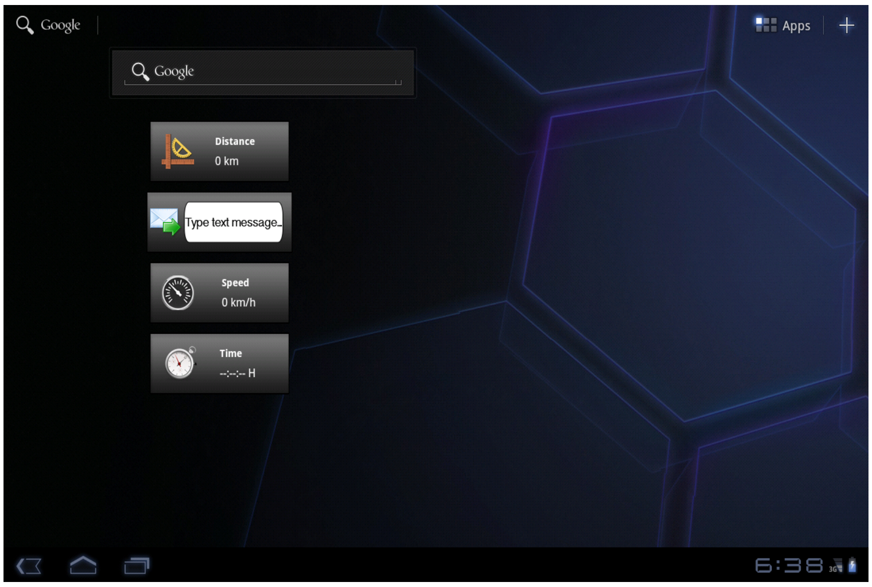


Figure 4.12: Pit unit home screen widgets

widget, a pending intent is sent and starts the activity, which the user can use to write and send the text message.

A pending intent is a token that is passed to a foreign application (e.g. Notification Manager, Alarm Manager, Home Screen AppWidget Manager, or other 3rd party applications), which allows the foreign application to use the application's permissions to execute a predefined piece of code. If the 3rd party application retrieves an intent object, and that application sends/broadcasts the intent, they will execute the intent with their own permissions. But if instead of an intent object, the application retrieves a pending intent, that application will execute the contained intent using the permissions of the application that has sent the pending intent.

DroidControl Live acquisition

Live acquisition is the user interface provided by the Droidcontrol application, in order to make possible the team members at the pit lane monitor the telemetry data of the HammerShark vehicle. The user interface is a single activity, which retrieves all information by the acquisition module via Intent objects. To view the produced data, the user must click on the Live acquisition button at the main page. After clicking, the configuration dialog (Figure 4.14(a)) is shown to the user.

The configuration dialog appears before starting the building of the user interface by the application and only advances if the user inserts valid information. The dialog is divided in three sub-sets, namely forward options, Log options and Track options. Forward options are

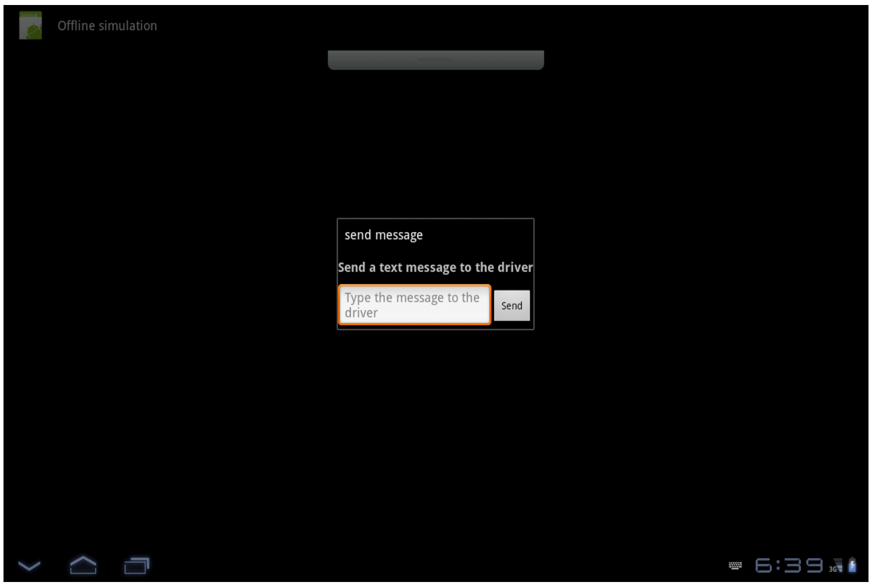

Offline simulation
send message
Send a text message to the driver
Type the message to the driver
Send
6:39

Figure 4.13: Opened activity after clicked on the message widget

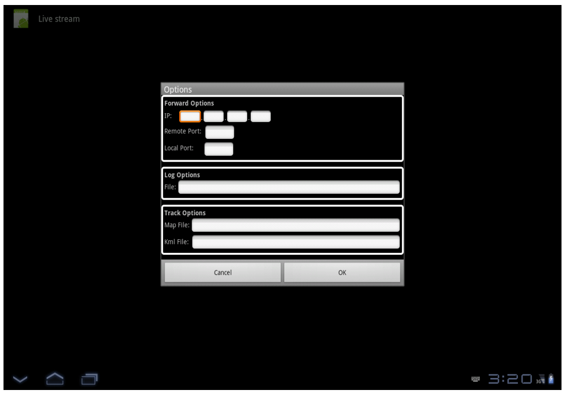
information that is necessary to establish a connection with the car unit, in order to enable the sending of text messages to the driver. Beside the connection, the user also indicates the port where the server is listening for incoming connections. The server port is important, because we could be in a network where not all ports are available and with this option we gave the ability to the user to choose an open port.

At the log option sub-set, the user indicates the name of the log files. The last sub-set is the track option. The track option serves for indicating the map file and the KML description of the track. This information is used by the applications to mark the vehicle's position on an offline map, similar to what occur at the car unit.

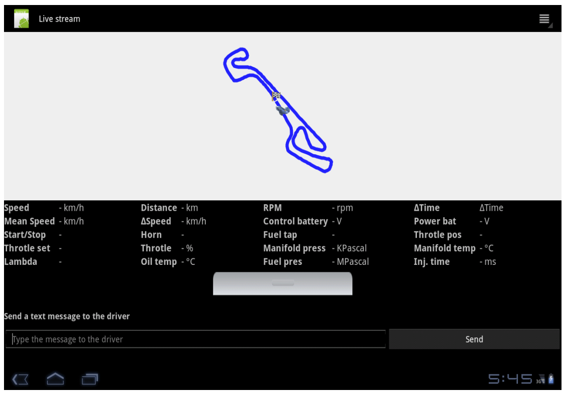
After the user has set all valid information at the dialog, the application loads the user interface (Figure 4.14(b)) and starts all background services that comprise the pit unit. At the top of the user interface is the track map with the position of the car at the track.The remaining space is filled up by all values that can be acquired from the HammerShark vehicle. Despite monitoring telemetry data, the user can also use the sliding drawer at the bottom of the user interface for sending text messages to the driver.

DroidControl Offline analysis

A good telemetry system is characterized not only by providing real time data monitoring, but also by allowing an offline analysis of the acquired data. For this purpose, we decided to offer the ability to analyze the whole competition at the pit unit. Since the pit unit is a tablet device and therefore fitted with a bigger screen, we could place all measured parameters in



(a) Live acquisition configuration dialog



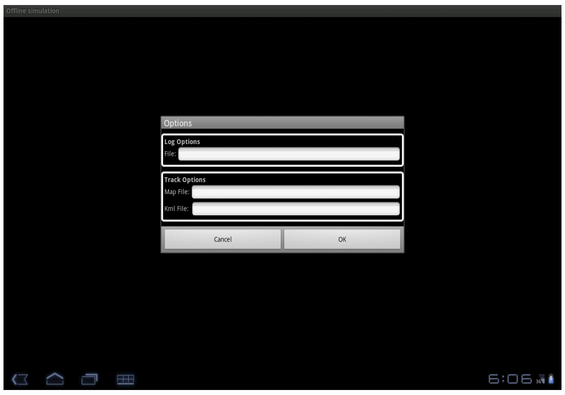
(b) Live aquisition UI

Figure 4.14: Pit unit live acquisition option

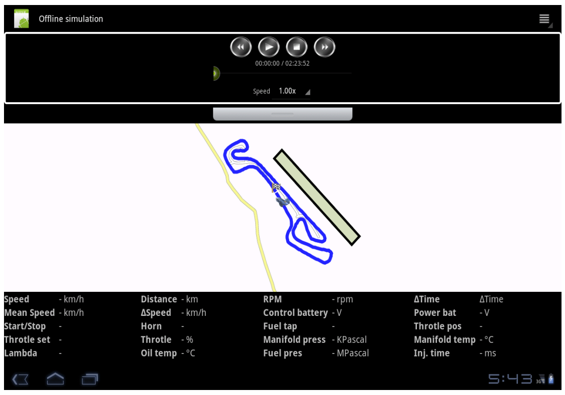
one single screen.

For accessing this feature, the user must choose the option Offline Analysis at the main screen that appears when the application is started. After choosing the option, the user is presented with a configuration dialog (Figure 4.15(a)), where he must select the offline map file location, the KML file with the track description, and also the log file that he wants to analyze. After validation of the selected files, the user can view an UI (Figure 4.15(b)) with the different measured parameters and the map of the track at the top.

For the offline analyses, we developed controls that are similar to well known media controls. These controls are hidden in a Sliding drawer on top of the activity, and offer the ability for starting, stopping and pause the simulation. The media controls also offers the ability to analyze the challenge in single step – forward using the forward button and backwards using the backward button.



(a) offline simulation configuration dialog



(b) offline simulation UI

Figure 4.15: Pit unit offline simulation option

4.4 Communication mechanism

In DroidShark we have the two types of communications, external communication for establishing a connection between both units and for telemetry data acquisition from the car, but also internal communication, in order to provide connectivity to the modules that comprises the applicaiton.

The communication between the different units – car unit and pit unit - is based on a simple protocol. To establish a communication we have several technologies as we have seen in subsection long-range communications of the chapter 2, but all these technologies only define the network. For data transmission we need to choose a transport protocol.

For the communication between the car unit and the pit unit we have a long- range communication, therefore we use Wi- fi in combination with the TCP protocol for establishing this connection. We choose Wi- fi because it's cheaper then using one of the other reviewed technologies in subsection long-range communications of the chapter 2 and we had facility to use. The only aspect where Wi- fi could be a concern, it's the range that is limited to hundred meters, in best scenario. In order to contour this disadvantage, we only upload data to the pit unit when the vehicle is in the range of the network. When the car is outside, no data is received at the pit lane.

The use of TCP contrary to UDP is due to the fact that TCP is a connection- oriented protocol. This means that a connection must be opened between the two end points. This is very useful for testing if the pit unit is in range of the network, without sending datagrams away without reaching the desination. TCP enables also data to be received in an ordered way, meaning if two packets are sent, then data packet 1 should be received before data packet 2. The order of the packets plays an important role at the pit unit, because of the data consistency that is shown to the users. Although a serial number or timestamp associated to each message, could ensure the same purpose.

Short- range communication is used for acquiring information from the vehicle. HammerShark vehicle has been developed for years and Bluetooth is already the protocol used to provide information from the CAN Bus. Beside this aspect, from all standards described in chapter 2, the best one would be Bluetooth. The smartphone is at the steering wheel and therefore we can't guarantee that it will be always in direct line of sight to the interface that provides information from the CAN Bus. Therefore, the use of IrDA couldn't work, as well as NFC. NFC work well if the distance is less than one meter. Such distance is too short to implement such mechanism on HammerShark. Remains Bluetooth, which beside from the long setup time seems to be the best solution to retrieve information from the car.

As refered above, DroidShark hasn't only external communications. The communication between the internal modules of the car unit and pit unit is an important aspect that needs attention. Based on the option using Android as mobile framework, we decided to support inter module communication on message passing, which is directly supported by the Android OS. To take advantage of this characteristic, we're using inter process communication mechanisms to establish communication between all modules. Therefore, we need an efficient inter process communication solution that ensures that DroidShark provides the information in time.

Android supports several inter process mechanisms, where the natural choice to support message passing is to use Intents. By using Intents, the modules that comprise the architecture could be loose coupled and the message passing management is delegating onto the Android OS.

4.5 Android IPC mechanism

Modern smartphones operating systems support the development of third- party applications with open system APIs. The Android Operating System provides beside an open API, rich inter- application collaboration that reduce developer burden by facilitating component reuse.

IPC is used at DroidShark for the communication between the modules that comprises the application (Figure 4.16). Before addressing DroidShark inter process communication structure, the Android's options must be known. Android supports several inter process mechanism, namely Intents [55], Android Interface Definition Language (AIDL) [38] and Content Providers [37].

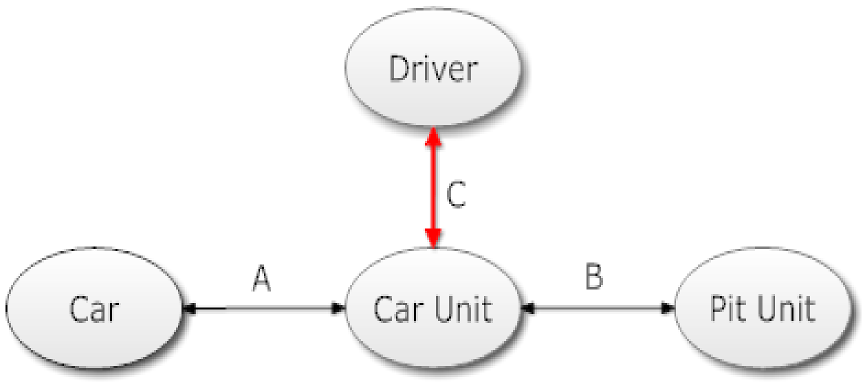


Figure 4.16: Internal communication in abstract overall scheme

4.5.1 AIDL

In Android there are two types of services, local services and remote services. Local services are services that are called only by the application that host them. Remote services are services that support a remote procedure call (RPC) mechanism. These services allow external clients, on the same device, to connect to the service and use its facilities.

To use RPC in Android, an interface definition language (IDL) is used to define the interface that will be exposed to clients. In Android this IDL is called Android Interface Definition Language (AIDL) [38] and is an IPC mechanism similar to COM or Corba [39]

Corba is characterized to integrate two different programs, written in different programming languages and running on different machines, into a single distributed application. Corba uses IDL to define how two programs communicate with each other. IDL has the advantage to be language- independent and therefore the solution to adopt when working with two programs, which are developed in different programming languages.

There are other inter process mechanism supported on high level programming languages, namely Java RMI [40] for the Java programming language. Java RMI is a Java application-programming interface that performs the object- oriented equivalent of Remote Procedure Call (RPC). Contrary to CORBA, Java RMI only supports making calls from one JVM to another. There is however, an important difference between CORBA and RMI. RMI requires the server classes to generate stubs and skeletons, not just the interfaces. Like CORBA, the generation of stubs and skeletons in AIDL requires only the defined interface. AIDL uses an aidl compiler to generate a Java interface definition, which must be made available to both the local and remote process, from an aidl file.

AIDL uses a proxy class to pass values between the client and the implementation [35]. How the Android generated classes fit together is depicted in figure 4.17.

To build a remote service, developers must first define an interface using AIDL. This interface definition is stored in a file with *.aidl* extension, where an aidl tool will then generate a Java interface. All remote methods must be implemented on the remote server and return the interface from the *onBind()* method.

4.5.2 Intents

Intent is the message passing mechanism of the Android OS. Intents are able to activate activities, services and broadcast receivers. The existing sending methods are described in Table 4.1. Beside activating component types, intents provide a facility for performing late runtime binding between the code in different applications, where the most significant use is the launching of activities.

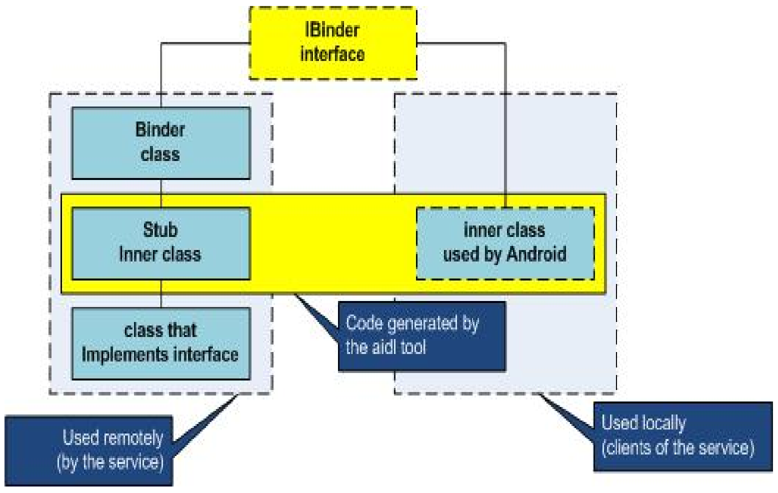

IBinder interface
Binder class
Stub Inner class
inner class used by Android
class that Implements interface
Code generated by the aidl tool
Used remotely (by the service)
Used locally (clients of the service)

Figure 4.17: Android AIDL inner classes[38]

An Intent is a passive data structure holding an abstract description of an operation to be performed. An Intent contains information of interest to the component that receives it, such as action to be taken and the data to act on. Despite this, an intent object holds also information of interest of the Android system, namely the category of component that should handle the intent.[55]

Intents can be used for explicit or implicit communication, where it defines a message to activate either specific component or a specific type of component. In other words, an explicit Intent identifies the intended recipient by name, whereas an implicit Intent leaves it up to the Android platform to determine which application(s) should receive the Intent. Using an explicit Intent guarantees that the Intent is delivered to the intended recipient, whereas implicit Intent allow for late runtime binding between different applications.

Intents are a powerful concept, with the following advantages:

- Loosely couples the application
- Can activate three core components of the android applications, namely activities, services, and broadcast receivers
- Using intents we can start any other activity that is present in the system and pass data between activities.

4.5.3 Content Provider

In order to manage the access to a structure set of data in Android OS, the Content Provider is used. A database is an example of a set of structured data that can be encapsulated

Receiver	sendBroadcast(Intent i)
	sendBroadcast(Intent i,String rcvrPermission)
	sendOrderedBroadcast(Intent i,String rcvrPermission,BroadcastReceiver receiver,...)
	sendOrderedBroadcast(Intent i,String recvrPermission)
	sendStickyBroadcast(Intent i)
	sendStickyOrderedBroadcast(Intent i,BroadcastReceiver receiver,. . .)

Activity	startActivity(Intent i)
	startActivityForResult(Intent i,int requestCode)

Service	startService(Intent i)
	bindService(Intent i,ServiceConnection conn,int flags)

Table 4.1: Intent sending methods signature

into a content provider. providing the standard interface for code running in another process access the database. [37]

Shared Content Providers can be queried for results, existing records updated or deleted, and new records added. Any application with the appropriate permissions can add, remove, or update data from any other application – including from the native Android databases. Many native databases are available as Content Providers, accessible by third- party applications, including the phone's contact manager, media store, and other native databases. To expose queries and transactions, the delete, insert, update and query method must be implemented on a Content Provider. These methods are the interface used by the Content Resolver to access the underlying data and if the most common scenario is to use a Content Provider to expose a private SQLite database, within these methods it's possible to access any source of data (including files or application instances). The queries in a Content Provider take a form very similar to that of database queries. Query results are returned as Cursors over a result set, like databases

Chapter 5

IPC Evaluation

The support of inter- process communication by the Android mobile operating system encourages the implementation of new software architectures with modular characteristics. Modular architectures may be implemented using services. Android services are designed to perform background processing even if the applications activities are stopped or invisible. Services are controlled from other components like other services, activities, and Broadcast Receivers.

As described in chapter 4, Android supports several inter process mechanisms, where the natural choice to support message passing is to use Intents. By using intents, the modules that comprise the architecture could be loose coupling and the message passing management is delegated onto the Android OS.

Therefore we used Intents to achieve inter module communication at the pit unit and at the car unit. This choice at the pit unit is emphasized by the use of home screen widgets for providing data visualization. Home screen widgets are mainly updated by using Intents and in order to follow the same strategy along the whole architecture, we decided to implement the pit unit using Intents as inter process mechanism.

After executing some preliminary tests at the car unit using Intents, the results were somewhat disappointing in relation to our initial expectations – the application become slow when we increased the acquisition rate. In this context, we decided to explore available mechanisms for IPC provided by the Android OS. To find out how far the results produced by Intents were bad, we decided to setup a evaluation to find out which mechanism has the better performance on our system and fits better for our needs, ensuring that our system can still deliver the information in time.

5.1 DroidShark ipc evaluation

We decided to perform a thorough evaluation in order to select the most suitable IPC method to use internally in DroidShark. For executing the evaluation, we used data collected by one previous race at the Nogaro racetrack in France. By using real data collected during a race, we are ensuring the proximity of the tests with the real environment. The tests cover a distance of equivalent of four laps, which is equivalent to a time of 59 minutes and 25 seconds. Each data frame has a size of 15 bytes and arrives at the smartphone with a rate of 4 data frames per second. In order to isolate the communications and message passing timings from other possible time consumer processes we considered several setups with different conditions:

- **Setup 1:** Only update the driver UI.
- **Setup 2:** Update the driver UI and perform data logging.
- **Setup 3:** Setup 2 and forward data to the pit unit.
- **Setup 4:** Setup 3 with internal GPS updates of every second.

For the evaluation of the IPC mechanism, we establish several measuring points at each internal modules (Figure 5.1). Measuring points have the objectives to discriminate internal IPC processing time and overall time spend in DroidShark from Car input to export information to the remote device. Each measuring point saves the time in timestamp format and provides enough information for obtaining the following times:

- **Ui-send:** Time needed by the acquisition module to send a message to the UI module.
- **Log-send:**Time needed by the acquisition module to send a message to the log module.
- **Gat-send:**Time needed by the acquisition module to send a message to the gateway module.
- **Ui-process:** Time that the UI module takes to update the UI with the received data.
- **Log-process:**Time that the log module takes to log the received data.
- **Gat-process:**Time that the gateway modules need to forward the received data to the pit unit.
- **Comm Time:** Time needed to disseminate data to all modules.
- **Process Time:**Time taken until all modules have processed a single message sent from the acquisition module.

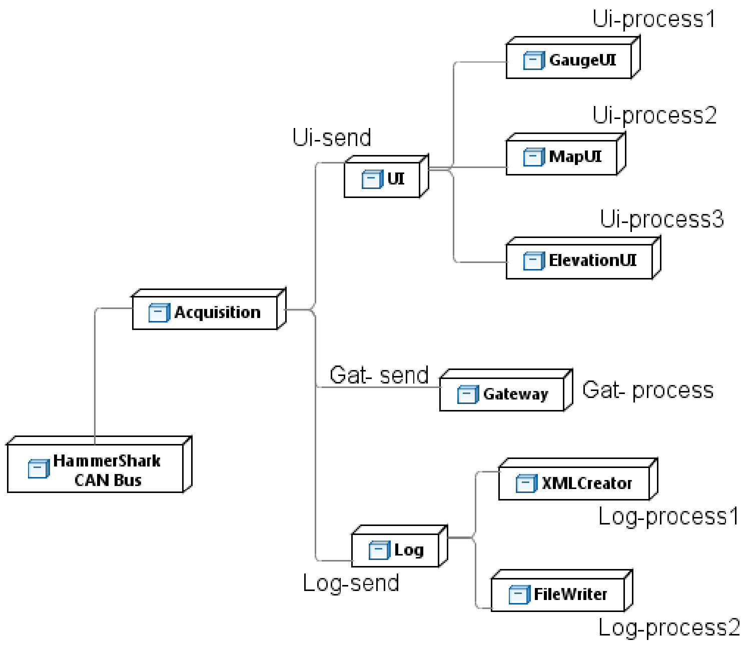


Figure 5.1: DroidShark internal architecture with the measuring points

These times are necessary to take conclusion about the performance and behaviour of each modules of DroidShark in specific scenarios. We performed 10 full runs, each run simulating an of the HammerShark in the track that consisted on 14300 messages, of 15 bytes each in a total of 209.47 kB of data transferred.

The smartphone used to perform the evaluations is a Samsung Galaxy S running on Android version 2.3.3 (Gingerbread) and the Android emulator with also the version 2.3.3 running as a replacement for the pit unit. To simulate the CAN bus of the vehicle, we used a laptop with a Bluetooth Server running that sends the streams to the phone.

5.1.1 IPC evaluation results

In order to take conclusions about the performance of both mechanisms during the evaluation, we centred our attention to the time needed by the acquisition module to disseminate data to all remaining modules (Comm Time) and the time that all modules needed to process one message (Process Time). The results of both measuring points are presented in Table 5.1 for the measuring point Comm Time and in Table 5.2 for the measuring point Process Time. On both tables we present the average time and the standard deviance for each setup. All values are calculated taking in consideration all data produced during the ten runs of each setup (14300 messages). The remaining tables with the results of the remaining measuring points can be found in Appendix A IPC evaluation results.

Average of the Comm Time					
		Setup 1	Setup 2	Setup 3	Setup 4
Intent	Mean(ms)	6.577	11.902	48.093	26.809
	Std	5.299	15.101	24.609	24.589
AIDL	Mean(ms)	0.472	36.890	39.652	42.371
	Std	0.815	11.157	18.930	20.358

Table 5.1: Comparison of both mechanisms for the Comm Time

Average of the Process Time					
		Setup 1	Setup 2	Setup 3	Setup 4
Intent	Mean(ms)	31.089	32.461	45.670	49.747
	Std	10.831	9.860	17.226	19.007
AIDL	Mean(ms)	33.952	40.031	43.545	46.963
	Std	12.809	10.292	17.180	18.200

Table 5.2: Comparison of both mechanisms for the Process Time

An overview over the results, show us that the performance of the AIDL mechanism depends on the number of modules that are running. If we take a closer look to the Comm Time in Table 5.1, we can observe that the mean time becomes worse as soon as we increased the number of services running. The same can also be observed when using Intents, however, setup 3 has worser performance than setup 4.

In order to find the reason for the worser performance of the setup 3 in comparison with setup 4, we analyzed with more detail the test results of both setups - setup 3 and setup 4. At the Figure 5.2 is depicted the comparison of both setups with the mean value of each executed runs.

The higher value observed in Table 5.1 for the full setup without gps updates in comparison with the full setup with gps updates is due to the results produced during the sixth run - outlier. The discrepancy of the values produced during the sixth run could be due to background tasks that the operating system has executed without knowledge of the user.

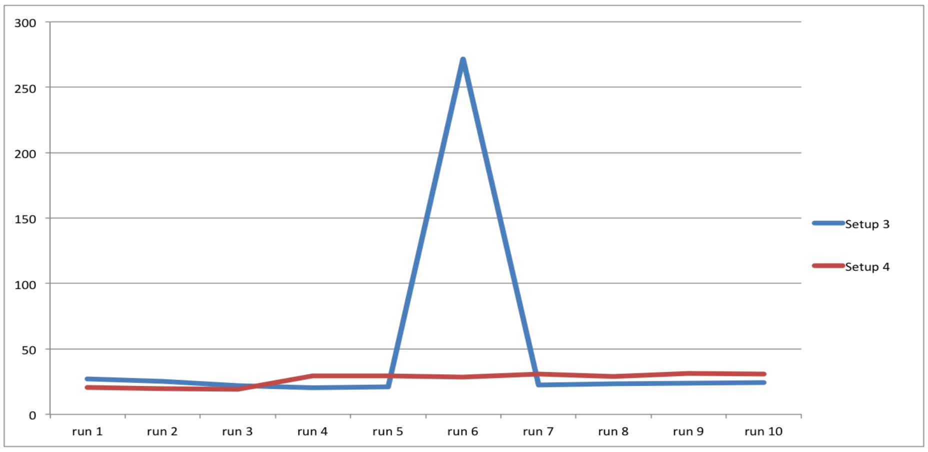


Figure 5.2: Comparison of the Full Setup without and with GPS using Intents

Average of the Comm Time					
		Setup 1	Setup 2	Setup 3	Setup 4
Intent	Mean(ms)	6.577	11.902	23.235	26.809
	Std	5.299	15.101	22.154	24.589
AIDL	Mean(ms)	0.472	36.890	39.652	42.371
	Std	0.815	11.157	18.930	20.358

Table 5.3: Comparison of both mechanisms for the Comm Time without the outlier

In order to the performance of both mechanisms, we removed the outlier (Table 5.3). Without the outlier, the behaviour of both mechanism was what we was expected. As soon as we added more background services or at the end the internal GPS updates, the results became worser. We can conclude that AIDL has better performance with a low number of background services than Intents, however we verify the opposite when we increase the number of background services running on background.

The better performance of the Intent mechanism in comparison to the AIDL mechanism, when we face a high number of services running in background was expected since it has to do with the implementation of each mechanism. The Intent mechanism is asynchronous. This means that the OS is responsible to disseminate the Intent for the several background services that are registered on the system. When the OS starts the data dissemination, all background services receives the intent almost at the same time, lowering so the time that the acquisition module needs to provide all modules with data. On other hand, the AIDL mechanism is synchronous, since it uses remote calls for providing the service with information. Therefore, when using AIDL, a background service only receives the data as soon as the execution flow of the application has returned from the previous background service.

We could observe a better performance of Intents at the measuring point Comm Time relative to AIDL. However, the same wasn't expected for measuring point Process Time. AIDL took less 22 seconds for concluding the evaluation and if it hasn't a good performance at the data dissemination, it must process the messages quicker than when using Intents. After observing Table 5.2, we observed similar results for both mecanisms. Therefore, in order to assert if the previous observations were statistically significant and that the observed trends translated a statistically valid conclusion, we decided to perform a statistical analysis.

5.1.2 Statistical analysis

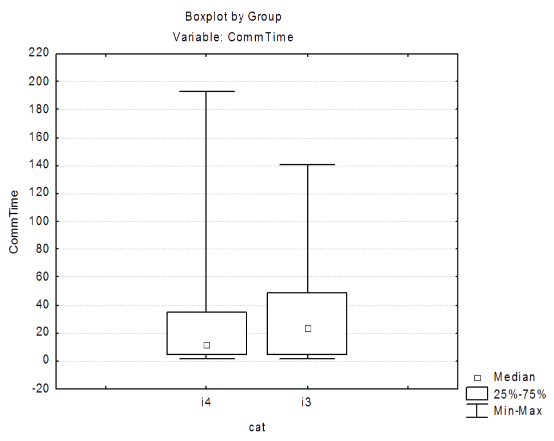
To perform the statistical analysis we used the Mann- Whitney U test [68] to test if the times measured for the setups belonged or not to the same distribution. In case of belonging to the same distribution we would discard that the observed trends represented significant differences between the setup.

The Mann- Whitney U test is a non- parametric statistical hypothesis test for assessing whether one of two samples of independent observations tends to have large values than the other. In our case, the values considered are the measured times for the considered setups. The test conditions were:

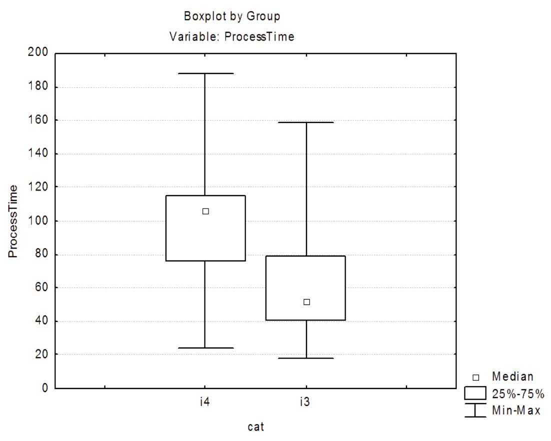
- The times measured using AIDL and Intents are from the same distribution, therefore, they don't differ statistically.
- Hypothesis: The time taken since data is sent from the acquisition module, until the last module receives it.
- if p <0.05 between the distributions of the time measured with AIDL and Intents, therefore we should consider AIDL and Intents from different distributions and statistically different.

For the first analysis, our focus was on comparing the Comm Time and Process Time between the considered conditions and verify if the measured times for each of the setups where statistically different. Due to limitations of the tool used to perform the analysis, we considered 10'000 messages (from a total of 14300 for each setup considered) to be able to perform a comparison between the setup with and without GPS updates using AIDL or Intents.

Our first comparison was between the full setup with GPS (Setup 4 - i4) and without GPS (Setup 3 - i3), when using Intents. We found a statistically relevance of p <0.05 meaning that there is a statistical difference between communication and processing time using Intents when the GPS is running simultaneously (setup 4) or not (setup 3).



(a) Elapsed communication time



(b) Elapsed processing time

Figure 5.3: Box plots for setup 3 and setup 4 using Intents

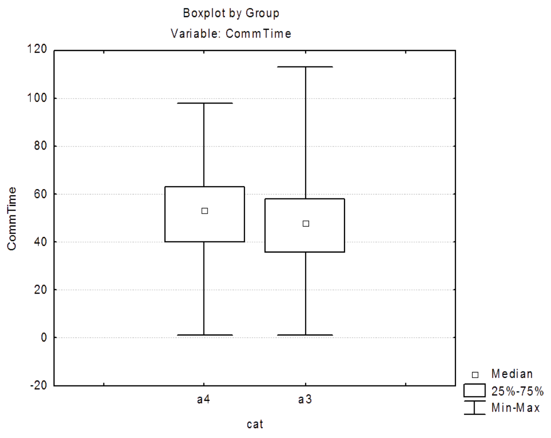
The acquisition module took less time, as depicted in figure 5.3(a), for disseminating the data to the remaining modules of the system when running in full setup with GPS updates (setup 4 - i4) in comparison with the full setup without GPS updates (setup 3 - i3). These results are a surprise, because we expected that with the use of the internal GPS, the system would respond slower. However, when we compared the setup 3 and the setup 4 for the time needed to process a message received by the acquisition module, as depicted in figure 5.3(b), the modules processed the data faster when running in setup 3.

After comparing the full setup (Setup 3) and the full setup including the internal GPS (Setup 4) using Intents, we analyzed the same setups, but using AIDL. Again we found a statistically relevance of $p < 0.05$, therefore there are also no statistical difference between communication and processing time using A when the GPS is running simultaneously (setup 4) or not (setup 3).

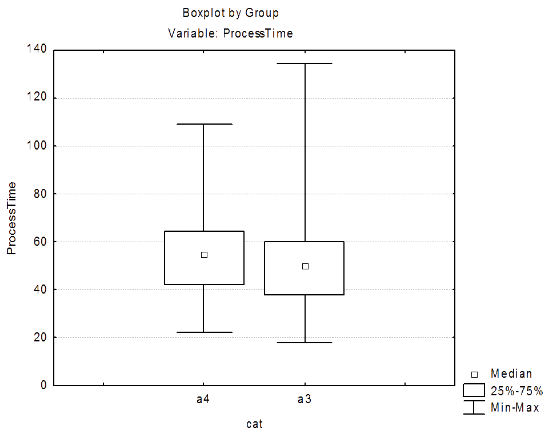
The produced figures with the statisitcal data showed us that the results of both setups were similar. This means that the use of the internal gps didn't affect too much the performance of the AIDL mechanism. As depicted in figure 5.4(a), the time needed by the acquisition module to disseminate the message to the reamaining modules is worse on the setup 4, but the results for setup 3 and setup 4 are very close together.

The time that all modules need to process the retrieved message is also better on setup 3, as depicted in figure 5.4(b). However, like the time needed to disseminate, also in this measuring point, the results of setup 3 and setup 4 were identical.

In order to take conclusions about the performance of both mechanisms, we performed



(a) Elapsed communication time



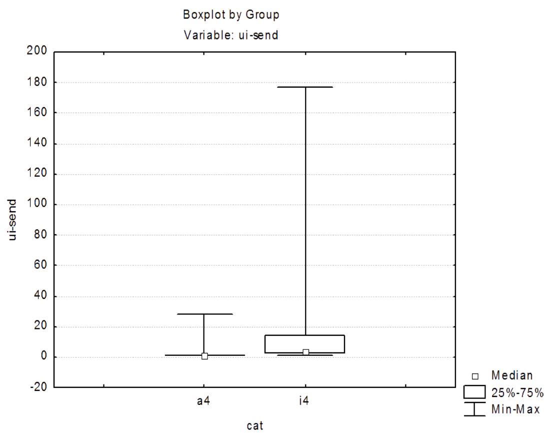
(b) Elapsed processing time

Figure 5.4: Box plots for setup 3 and setup 4 using Intents

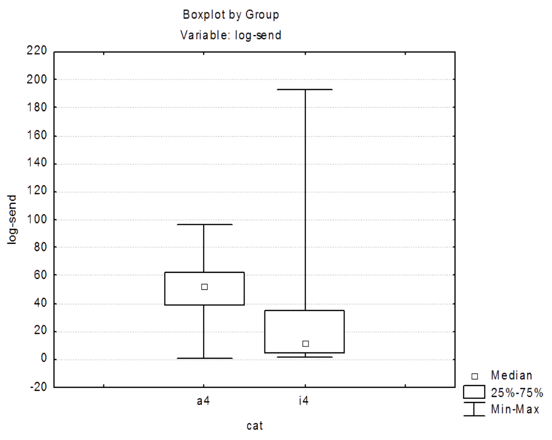
a direct comparison of the full setup with GPS updates with both mechanisms. Beside the times that serve for comparison on the previous analysis - Comm Time and Process Time - we compared also the following times:

- **Ui-send:** Time elapsed since the data is sent from the acquisition module, until it arrives at the UI module.
- **Log-send:** Time elapsed since the data is sent from the acquisition module, until the log module receives it.
- **Gat-send:** Time elapsed since the data is sent from the acquisition module, unitl it arrives at the gateway module.

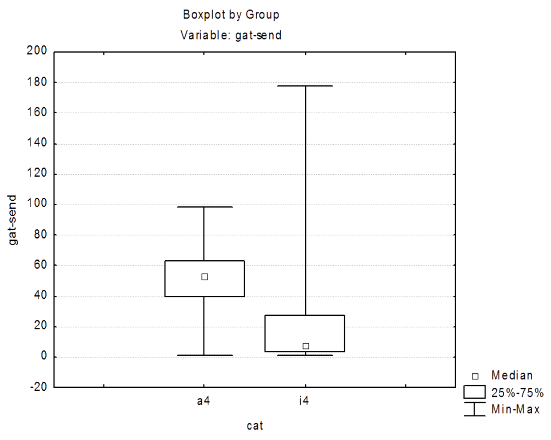
After the analysis we found p <0.05, meaning that there is a statistical difference between the distributions of the time measured with AIDL and intents, therefore we should consider AIDL and intents from different distributions and statistically different. The results show us also that the performance of the mechanisms depends on the number of background services that we are providing with information.



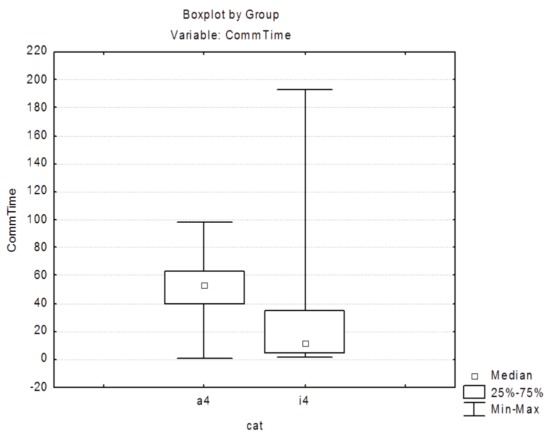
(a) Comparison of Ui-send on both mechanisms



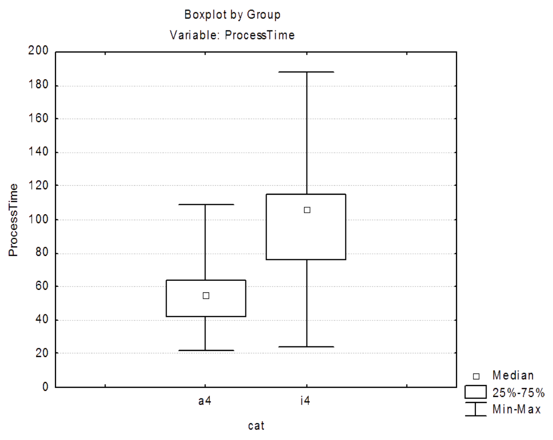
(b) Comparison of Log-send on both mechanisms



(c) Comparison of Gat-send on both mechanisms



(d) Comparison of Comm Time on both mechanisms



(e) Comparison of Process Time on both mechanisms

Figure 5.5: Box plots with the comparison of the performance of AIDL with Intent running in setup 4

As we can see in figure 5.5(a), AIDL is faster in providing the UI module with data than Intents. However, the same didn't occur with the Log module (figure 5.5(b)) and with the Gateway module (figure 5.5(c)). The reason is that both modules – Log module and Gateway module – are only provided with information after the UI module has been provided with data and has concluded the processing of the received message. This happens, as we saw above, because AIDL is a synchronous mechanism. Intents are asynchronous and therefore all modules can be provided with data at the same time. This result in being faster in providing all modules with data as the AIDL mechanism (figure 5.5(d)).

Providing all modules with data at the same time is faster at the communication level, however, that could become a problem when each module needs CPU time for processing the retrieved message. As expected, modules take less time for processing data when they are provided by AIDL than by Intents. Since Intents are asynchronous, the CPU time is divided and each module only gets a small amount of CPU time for processing the received message. When using the AIDL mechanism, a new module only receives a message, when the previous module has finished to process the message. Therefore, the amount of CPU time that is available for each module is higher.

At Table 5.4 we have the total time needed by each mechanism to execute the evaluation under the several conditions. AIDL has executed the evaluation faster than Intent for each setup, except for the full setup with GPS updates (setup 4), where both mechanisms required the same time for executing the test.

The tables with all results of the statistical analysis can be found in appendix B Mann-Whitney U Test Results.

Total times for each tested setup				
	Setup 1	Setup 2	Setup 3	Setup 4
Intent	9:54:07 H	9:54:00 H	9:54:10 H	9:53:51 H
AIDL	9:54:04 H	9:53:59 H	9:53:52 H	9:53:51 H

Table 5.4: Total time of each setup with both mechanisms

5.1.3 IPC evaluation with increased transfer rate

After executing runs with a transfer rate of 4 streams per second, which is the transfer rate that is used by the HammerShark vehicle to provide DroidShark with information, we decide to increase the transfer rate in order to observe the behaviour of the system. We decide to execute one run with a transfer rate of 8 streams per second and another run with a transfer rate of 12 streams per second. The conditions and measuring points are the same of the previous evaluation.

Contrary to expectations, as we can see on Table 5.5, the performance of the system was better with a transfer rate of 12 streams per second than with 8 streams per second. Even when we compare with the results that we obtained with a rate of 4 streams per second, we achieved a better performance with higher rates. Neverthless, the tests confirms the fact that AIDL becomes worse when we increase the number of modules that are running and the fact that the modules take longer to process the received messages when using Intents as IPC mechanism.

We noticed during the analysis of the produced data, that when using Intents with a transfer rate of 8 streams per second, 360 streams of the 14300 sent don't were received by the modules. With a transfer rate of 12 streams per second, we had 448 streams of the 14300 that were not sent to the modules. This results in a loss of 2.52 % of the streams with a transfer rate of 8 streams per second and a loss of 3.13% of all streams when the transfer rate was 12 streams per second during the evaluation. Contrary to Intents, the AIDL mechanism had delivered all sent streams to the modules.

	8 streams/second				**12 streams/second**			
	AIDL		Intent		AIDL		Intent	
	Mean (ms)	Std	Mean (ms)	Std	Mean (ms)	Std	Mean (ms)	Std
Average Ui-send	0.297	0.948	5.935	5.022	0.151	0.550	5.401	12.687
Average Log-send	13.682	21.684	5.718	32.990	3.499	12.705	5.115	23.530
Average Gat-send	14.874	22.364	4.876	26.261	4.788	14.715	4.729	20.042
Average Ui-process	35.853	15.158	34.100	14.377	27.706	17.083	25.419	18.546
Average Log-process	45.338	16.627	55.163	20.893	56.654	19.730	55.891	19.050
Average Gat-process	6.733	15.724	1.324	1.389	5.100	14.381	1.145	1.487
Average Comm Time	14.874	22.364	10.591	11.995	4.788	14.716	8.386	7.819
Average Process Time	50.276	13.947	57.101	19.583	58.125	18.786	56.943	18.734

Table 5.5: Table with the results of the evaluation with incrased transfer rate

In order to find out the causes that led to data loss when the communication was made using Intents, we tried to find out how the OS handles broadcasted intents. As referred earlier in this chapter, Intents are only data structures. Therefore, the OS needs to know what to do with it. The handling of the Intents is done by the process Intent resolution, which maps an Intent to an Activity, BroadcastReceiver, or Service that can handle it. The intent resolution mechanism basically resolves around matching an Intent against all of the Intent-filter descriptions in the installed application packages. There are three pieces of information on every Intents that are used for resolution: the action, the type, and category. Using this information, a query is done at the PackageManager for a component that can handle the intent. The appropriate component is determined based on the intent information supplied in the AndroidManifest.xml file. The whole procedure is depicted in figure 5.6.

When we increase the transfer rate, the broadcastreceiver that handles an Intent of this type can be busy for a certain time period, forcing the system to wait with sending the intent until the service is newly available for receiving intents. When the system receives a new intent during the delay time, which match the three pieces of information described above – action, type and category – the system considers this to be a duplicate and therefore the old one will be discarded.

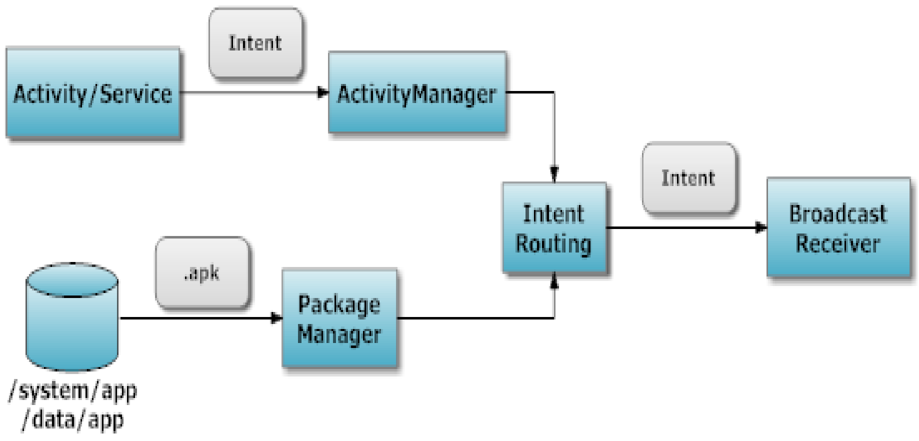


Figure 5.6: Intent handling overview

5.2 Final Considerations

After a careful analysis of the results produced by the evaluation, we decided to implement AIDL as the IPC mechanism of the car unit. AIDL took less 22 seconds for executing the evaluation and the modules required less time to process data. Both aspects were decisive for the choice of AIDL as the IPC mechanism. Despite the better performance shown on our evaluation, AIDL also offers a reliable method for providing data to all background services that comprises the architecture, even when the transmission rate was increased.

Even with all this advantages of the AIDL mechanism at our application, we decided not to change the implementation of the pit unit from Intents to AIDL. This decision is based on the update method for home screen widgets. Home screen widgets are mainly updated by Intents and led us to implement Intents as the IPC mechanism at the pit unit. By implementing Intents as the IPC mechanism at the pit unit, we have a uniform mechanism, which is transversal to the whole DroidControl application.

Chapter 6

Conclusions and Future Work

We presented a telemetry system called DroidShark with the aim of supporting in- race strategies at the Shell Eco- Marathon. We researched several telemetry systems that are used in moto sport events and solutions for the regular car driver. For motor sport events we don't find any solution based on mobile operating systems. They are mainly custom hardware with the specific task of logging data or forward data to a central unit, which stays mainly at the pit lane. With such solution we can obtain high data transfer rate, because there is no concern on providing visualization for the driver by the system. However, solutions that we found for the regular car driver are mobile operating system based. The use of the smartphone dispenses the existence of custom hardware, turning the system more convenient for the end user.

DroidShark is an Android OS based telemetry system, implemented using a modular architecture allowing us to reuse components between the two main nodes of the DroidShark system - the car unit and the pit unit. For that reason our focus throughout the dissertation was given on the IPC between the components of the system.

The Android OS provides several IPC mechanisms that could be used. In order to select the one that fits better in our system, we performed a selection process for the Android IPC to be used in our system. During this process we executed an evaluation, where we achieved interesting results that, in our opinion, could be helpful for Android application development in other scopes. The evaluation has shown that it is worth to analyze the several IPC mechanisms in the application scenario, because the right choice can improve the performance of the application. In the concrete case of DroidShark, we select the AIDL mechanism. As discussed in detail in the previous chapter, it provides the best option in this scope. AIDL ensures better reliability in comparison with Intents, namely in stable communication times, better overall performance and had no data loss when we increased the transfer rate.

The DroidShark architecture implies external communication between units. No concrete evaluation was performed between the serveral option found and reviewed. This was due to the focus of the dissertation being in the telemetry system components and to the added complexity of including, besides the connectivity related issues, monetary cost namely to evaluate cellular networks.

6.1 Future work

As future work we have to perform a real test with our telemetry system. Until now, we have only tested the telemetry system in a simulated environment. Even if we have used real data from a previous race, the test is important in order to proof that the telemetry system has no issues that must be fixed. Beside from testing the telemetry system on a real environment, testing the UI design is also part of the future work. The UI design test would show us if the adopted UI is ideal for the driver or if we should change the layout. We have already some improvements that can be integrated in DroidShark. The switch to a cellular network of the long- range communication is on top of these improvements. The use of Wi- fi is problematic on a real environment due to the limited range of 100 meters and since it uses burst transmission, the pit unit doesn't receive all data in time for providing live monitoring. A cellular network would provide a persistent communication during the whole race, rather than only when the vehicle is in the range of the network. With these kind of networks, the live monitoring of the vehicle parameters during the race would not be a problem anymore. Beside switching the communication technology, another improvement would be to provide the ability to the team members at the pit to follow the race via video. The vehicle would be fit with a camera and DroidShark would be responsible to forward the information gathered from the camera to the pit unit.

Appendix A

IPC evaluation results

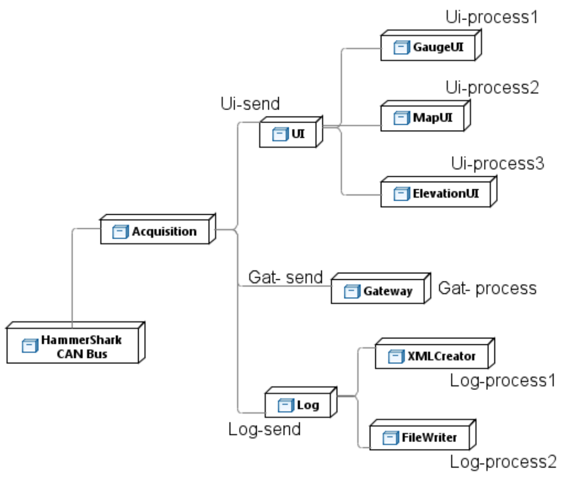


Figure A.1: DroidShark internal architecture with the measuring points

Average of Ui-send					
		Setup 1	Setup 2	Setup 3	Setup 4
Intent	Mean(ms)	6.577	5.844	32.697	8.757
	Std	5.299	5.478	10.599	6.979
AIDL	Mean(ms)	0.472	0.477	0.618	0.562
	Std	0.815	0.236	1.288	1.049

Table A.1: Comparison of both mechanisms for Ui-send with the outlier

Average of Ui-send					
		Setup 1	Setup 2	Setup 3	Setup 4
Intent	Mean(ms)	6.577	5.844	7.792	8.757
	Std	5.299	5.478	7.112	6.979
AIDL	Mean(ms)	0.472	0.477	0.618	0.562
	Std	0.815	0.236	1.288	1.049

Table A.2: Comparison of both mechanisms for Ui-send without the outlier

Average of Log-send					
		Setup 1	Setup 2	Setup 3	Setup 4
Intent	Mean(ms)	N/A[1]	11.902	48.093	26.809
	Std	N/A[1]	15.100	24.609	24.589
AIDL	Mean(ms)	N/A[1]	36.890	39.063	41.701
	Std	N/A[1]	11.157	18.850	20.351

Table A.3: Comparison of both mechanisms for Log-send with the outlier

Average of Log-send					
		Setup 1	Setup 2	Setup 3	Setup 4
Intent	Mean(ms)	N/A[1]	11.902	23.232	26.809
	Std	N/A[1]	15.100	22.154	24.589
AIDL	Mean(ms)	N/A[1]	36.890	39.063	41.701
	Std	N/A[1]	11.157	18.850	20.351

Table A.4: Comparison of both mechanisms for Log-send without the outlier

[1]N/A – Module not running on this setup

Average of Gat-send					
		Setup 1	Setup 2	Setup 3	Setup 4
Intent	Mean(ms)	N/A[1]	N/A[1]	41.115	18.824
	Std	N/A[1]	N/A[1]	20.515	20.720
AIDL	Mean(ms)	N/A[1]	N/A[1]	39.652	42.371
	Std	N/A[1]	N/A[1]	18.930	20.358

Table A.5: Comparison of both mechanisms for Gat-send with the outlier

Average of Gat-send					
		Setup 1	Setup 2	Setup 3	Setup 4
Intent	Mean(ms)	N/A[1]	N/A[1]	16.245	18.824
	Std	N/A[1]	N/A[1]	17.790	20.720
AIDL	Mean(ms)	N/A[1]	N/A[1]	39.652	42.371
	Std	N/A[1]	N/A[1]	18.930	20.358

Table A.6: Comparison of both mechanisms for Gat-send without the outlier

Average of Ui-process					
		Setup 1	Setup 2	Setup 3	Setup 4
Intent	Mean(ms)	31.089	32.461	45.670	49.747
	Std	10.831	9.860	17.226	19.007
AIDL	Mean(ms)	33.952	40.031	44.124	47.478
	Std	12.809	10.292	17.348	18.345

Table A.7: Comparison of both mechanisms for Ui-process

[1]N/A – Module not running on this setup

Average of Log-process					
		Setup 1	Setup 2	Setup 3	Setup 4
Intent	Mean(ms)	N/A[1]	2.525	2.588	1.569
	Std	N/A[1]	13.281	16.651	10.762
AIDL	Mean(ms)	N/A[1]	0.439	0.524	0.577
	Std	N/A[1]	2.870	3.725	4.110

Table A.8: Comparison of both mechanisms for Log-process

Average of Gat-process					
		Setup 1	Setup 2	Setup 3	Setup 4
Intent	Mean(ms)	N/A[1]	N/A[1]	2.014	2.498
	Std	N/A[1]	N/A[1]	2.854	3.664
AIDL	Mean(ms)	N/A[1]	N/A[1]	1.601	2.104
	Std	N/A[1]	N/A[1]	6.857	7.695

Table A.9: Comparison of both mechanisms for Gat-process

[1]N/A – Module not running on this setup

Appendix B

Mann- Whitney U Test Results

	Max. Negative	**Max. Positive**	**U**	**p-level**	**Valid N**
CommTime	-0.153400	0.114700	43859571	0.00	10000
ProcessTime	0.000000	0.531200	17888680	0.00	10000

Table B.1: Mann- Whitney U Test results of comparison of the Full setup without and with GPS update using Intents

	Max. Negative	**Max. Positive**	**U**	**p-level**	**Valid N**
CommTime	-0.026500	0.109100	44481377	0.000000	10000
ProcessTime	-0.036800	0.0099300	44748180	0.000000	10000

Table B.2: Mann- Whitney U Test results of comparison of the Full setup without and with GPS update using AIDL

	Max. Negative	**Max. Positive**	**U**	**p-level**	**Valid N**
Ui- send	-0.968400	0.000000	1233036	0.000000	10000
Log- send	-0.062600	0.490400	24586524	0.000000	10000
Gat- send	-0.057500	0.617200	16456286	0.000000	10000
CommTime	-0.057800	0.494200	23942799	0.000000	10000
ProcessTime	-0.707400	0.000300	9183979	0.000000	10000

Table B.3: Mann- Whitney U Test with comparison of Intents with AIDL using the Full setup with GPS updates

References

[1] Shell Eco- Marathon, `http://www.shell.com/home/content/ecomarathon/`, October 2011.

[2] Icaro, `http://icaro.ua.pt/`, September 2011.

[3] Android OS, `http://www.android.com/`, September 2011.

[4] Google Play, `https://play.google.com/store?hl=en`, May 2012.

[5] Torque Pro, `http://torque-bhp.com/`, October 2011.

[6] Daimler A.G, `http://media.daimler.com/dcmedia/`, October 2011.

[7] AutoBild.de, `http://www.autobild.de/artikel/amg-performance-media-paket-\ \premiere-im-sls-amg-1943835.html`, October 2011.

[8] rpmgo.com, `http://www.rpmgo.com/amg-performance-media-mercedes`, November 2011.

[9] DTU Roadrunners, `http://www.ecocar.mek.dtu.dk/Innovator.aspx`, April 2012.

[10] F. Kneisler, N. Dummer, “Sensorik und Telemetrie für ein Brennstoffzellenfahrzeug mit Embedded Linux, TCP/IP und WLAN”, Hamburg University of Applied sciences, 2008.

[11] E. P. Alfaro, “Telemetry System of a Competition Car”, University of Pontificia Comillas, 2006

[12] The Economic Times, `http://economictimes.indiatimes.com/pictures/videos/pictures/formula-1-india-how-money-flows-in-worlds-most-expensive-sport/articleshowpics/10536557.cms`, May 2012.

[13] F. Heuing, F. Taterka, J. Hesselmann, “Connected Cars in der Formel 1”, University of Applied Sciences for Economics and Management in Düsseldorf, 2010

[14] Magneti Marelli, http://www.magnetimarelli.com/english/motorsport.php, May 2012.

[15] K. Pazul, “Controller Area Network (CAN) Basics”, Microchip Technology Inc., 2002.

[16] CAN Bus, http://www.canbuskit.com/what.php, March 2012.

[17] Wi-fi Alliance, https://www.wi-fi.org/discover-and-learn, May 2012.

[18] W. Lemstra, V. Hayes, J. Groenwegen, ”The Innovation Journey of Wi-Fi: The Road To Global Success”, Cambridge University Press, 2010,

[19] Bluetooth, https://www.bluetooth.org/, May 2012.

[20] Bluetooth SIG Inc., ”Specification of the Bluetooth System”, Bluetoot SIG Inc.,2003

[21] K. Phanse, J. Nykvist, ”Opportunistic Wireless Access Networks”, Proceeding AcessNets ’06 Proceedings of the 1st international conference on Access networks Article No. 11

[22] Ludwig R., Rathnoyi B., ”Link layer enhancements for TCP/IP over GSM”, INFOCOM ’99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings, IEEE, 21-25 Mar 1999

[23] ETSI, http://www.etsi.org/WebSite/Technologies/gsm.aspx, May 2012.

[24] 3GPP, http://www.3gpp.org/ftp/Specs/html-info/45-series.htm, May 2012.

[25] 3GPP, http://www.3gpp.org/ftp/Specs/html-info/25-series.htm, May 2012.

[26] Dr. Faris Muhammad, ”An Introduction to UMTS Technology: Testing, Specifications and Standard Bodies for Engineers and Managers”, BrownWalker Press, 2008

[27] ETSI, http://www.etsi.org/WebSite/Technologies/UMTS.aspx, May 2012.

[28] Motorola, ”Long Term Evolution (LTE): A Technical Overview”, Motorola, July 2010.

[29] Motorola, ”Long Term Evolution (LTE)”, Motorola, April 2011.

[30] 3GPP LTE Encyclopedia, https://sites.google.com/site/lteencyclopedia/home, July 2012.

[31] ETSI, http://www.etsi.org/WebSite/Technologies/lte.aspx, May 2012.

[32] 3GPP, http://www.3gpp.org/LTE/, May 2012.

[33] R. Wadham, ”Radio Frequency Identification”, Library Mosaics v14 n 5 pg. 22.

[34] Ericsson, "Scatternet - Part 1 Baseband vs. Host Stack Implementation", White paper, Ericsson Technology, 2004

[35] E.Chin, A. Felt, K. Greenwood, D. Wagner, "Analyzing Inter- Application Communication in Android" in MobiSys '11 Proceedings of the 9th international conference on Mobile systems, applications, and services, 2011.

[36] N. Preezm, QuickStart Guide to Developing AndroidTM Applications, Procwave, pp. 145 - 146, 2010.

[37] S. Komatineni, D. MacLean, S. Hashimi, "Pro Android 3", Apress, 2011.

[38] M. Murphy, "The Busy Coder's Guide to Android Development", CommonsWare, 2011.

[39] M. Henning, "The Rise and Fall of CORBA",ACM Queue 4, 2006.

[40] Oracle, `http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html`, July 2012.

[41] Android Beam, `http://developer.android.com/guide/topics/nfc/nfc.html#p2p`, May 2012.

[42] R. Thomas, M. Zehrfeldt, "IP-basierte Fernwirknetze über GSM, UMTS, WLAN, Wimax und Tetra", etz- Elektronik + Automation, 2008.

[43] Pankaj Bhatt "Bluetooth", Technology Review 2000-3, TATA Consultancy Services, 2000.

[44] P. Nielsen, "Introduction of the third generation of mobile communication system in Norway", Department of informatics, University of Oslo, 2002.

[45] J. Doyle, "CCIE Professional Development Routing TCP/IP", Volume 1, Cisco Press, 2005

[46] G. A. Abed, M. Ismail, and K. Jumari, "Behavior of cwnd for TCP source variants over parameters of LTE networks," Information Technology Journal, vol. 10.

[47] S. Steinke, "TCP And UDP", Network Magazine, 2001.

[48] A. Alexiou, C. Bouras, V. Igglesis, "Performance Evaluation of TCP over UMTS Transport Channels", 7th International Symposium on Communications Interworking (INTERWORKING 2004), Ottawa, Canada, November 29, 30 - December 1 2004.

[49] LTE Frequency Bands, http://www.3glteinfo.com/lte-and-lte-advanced-frequency-bands-and-spectrum-aloactions-20120224/, May 2012.

[50] IrDA, http://irdajp.info/irsimple.html, June 2012.

[51] Charles D. Knutson, "Infrared Data Communications with IrDA: A Tutorial." Proceedings of the 2002 Embedded Systems Conference, San Francisco, California, March 12-16, 2002.

[52] E. Ferro and F. Potortì, "Bluetooth and WiFi wireless protocols: a survey and a comparison", IEEE Wireless Communications, Vol. 12, No. 1, pp. 12-26, february 2005.

[53] F. Bennett, D. Clarke, J. Evans, A. Hopper, A. Jones, D. Leask, "Piconet Embedded Mobile Networking", Olivetti Oracle Res. Lab., Cambridge , 1997.

[54] A. Paus, "Near Field Communication in Cell Phones", University of Bochum, 2007.

[55] Intents and Intent Filters, http://developer.android.com/guide/components/intents-filters.html, July 2012.

[56] R. Meier, "Professional Android 2 application development", Wiley- India, 2010.

[57] Open Handset Alliance, http//www.openhandsetalliance.com, May 2012

[58] Mapsforge, http://code.google.com/p/mapsforge/, February 2012.

[59] AndroidPlot, http://androidplot.com/, February 2012.

[60] OpenStreetMap, http://www.openstreetmap.org/, February 2012.

[61] Google developers, https://developers.google.com/kml/documentation/kmlreference?hl=de-DE , May 2012.

[62] RotateAnimation, http://developer.android.com/reference/android/view/animation/RotateAnimation.html, October 2011.

[63] Map View, http://developer.android.com/resources/tutorials/views/hello-mapview.html, February 2012.

[64] Google Maps API, https://developers.google.com/maps/?hl=en, May 2012.

[65] Google Earth, http://www.google.com/earth/index.html, May 2012.

[66] IBM, http://www.ibm.com/developerworks/webservices/library/ws-restful/, July 2012.

[67] SAXParser, `http://developer.android.com/reference/javax/xml/parsers/SAXParser.html`, March 2012.

[68] Mann, Henry B., Whitney, Donald R., "On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other". Annals of Mathematical Statistics 18,(1947).50–60.