

**Project no.:** IST-FP6-STREP- 26979  
**Project full title:** Highly dependable IP-based networks and services  
**Project Acronym:** HIDENETS  
**Deliverable nr:** D6.3  
**Title of the deliverable:** Experimental proof-of-concept set-up  
**HIDENETS**  
 (Description of HIDENETS test-bed implementation)

<b>Contractual Date of Delivery to the CEC:</b>	August 31 <sup>st</sup> 2008
<b>Actual Date of Delivery to the CEC:</b>	August 29 <sup>th</sup> 2008
<b>Organisation name of lead contractor for this deliverable:</b>	WMC
<b>Author(s)/Participant(s):</b>	Manfred Reitenspieß (editor), Irene de Bruin, António Casimiro, Mario Calha, Zoltan Egel, Geir Egeland, Lorenzo Falai, Bjarke Freund-Hansen, Sonia Heemstra de Groot, Audun Fossellie Hansen, Gábor Huszerl, Marc-Olivier Killijian, András Kövi, Tom Lippmann, Luis Marques, Erling V. Matthiesen, Anders Nickelsen, Gergely Pintér, Matthieu Roy, Hans-Peter Schwefel, Gaëtan Séverac, Inge-Einar Svinnset, Christophe Zanon
<b>Work package contributing to the deliverable:</b>	WP6
<b>Nature:</b>	R
<b>Version:</b>	1.0
<b>Total number of pages:</b>	54
<b>Start date of project:</b>	1 <sup>st</sup> Jan. 2006
<b>Duration:</b>	36 months

**Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)**

Dissemination Level		
<b>PU</b>	Public	<b>x</b>
<b>PP</b>	Restricted to other programme participants (including the Commission Services)	
<b>RE</b>	Restricted to a group specified by the consortium (including the Commission Services)	
<b>CO</b>	Confidential, only for members of the consortium (including the Commission Services)	

**Abstract:**

This document contains refined descriptions of the HIDENETS test-bed implementations, which are used to show the feasibility of selected parts of the HIDENETS solutions when implemented on COTS components and to assess their impact on dependability and performance in an experimental laboratory environment. In order to focus on selected essential HIDENETS functionalities and to show their benefits when applied to specific applications and scenarios, four focus areas have been identified which lead to the definition of the following four test-beds:

- A complex resilient application developed on top of the HIDENETS solutions using model-based development methods. It involves both the infrastructure and the ad-hoc domain. Moreover, it illustrates the feasibility of using HIDENETS concepts, methods and middleware for high availability applications, like SAForum [8] Application Interface Specification does over the fixed infrastructure. The interoperability of the ad-hoc and fixed infrastructure is illustrated as well.
- A platooning application that is used as a proof-of-concept for the ability to detect and react to timing faults, to assure safety and to handle certain malicious intrusions.
- A distributed black-box application with the crucial middleware functionality that provides major dependability benefits in this application setting.
- Resilient communication protocols for ad-hoc (car to car) networks and their impact on higher-layers.

Each of the four test-beds can be run in a stand-alone set-up, allowing for parallel development and testing. Together, they represent a collection of integrated components and cover all relevant HIDENETS aspects and can later be used to show additional benefits from various combinations of HIDENETS middleware blocks.

As one of the major characteristics of the HIDENETS scenarios is the dynamicity of the communication link properties and of the topologies in the ad-hoc domain, also a topology emulation tool is described. With this tool, reproducible experiments with dynamic topologies can be executed in all four test-beds or in combinations thereof.

**Keyword list:** HIDENETS, dependability, communication systems, ad-hoc networks, wireless, middleware solutions, car-to-car communication, applications, COTS components, test-beds.

# Table of Contents

<b>1</b>	<b>EXECUTIVE SUMMARY .....</b>	<b>5</b>
<b>2</b>	<b>INTRODUCTION .....</b>	<b>7</b>
<b>3</b>	<b>APPLICATION DEVELOPMENT TEST-BED .....</b>	<b>11</b>
3.1	INTRODUCTION.....	11
3.2	REQUIREMENTS CONSIDERED FOR THE TEST-BED.....	11
3.2.1	<i>Use-case requirements</i> .....	11
3.2.2	<i>HW/SW requirements</i> .....	12
3.3	TECHNICAL DETAILS OF THE TEST-BED.....	13
3.3.1	<i>Applied middleware components</i> .....	13
3.3.1.1	Application of the Hidenets middleware .....	14
3.3.1.2	Application of a SAF-AIS Compliant Middleware .....	14
3.3.1.3	A short summary of the High-Availability (HA) oriented approach .....	14
3.3.2	<i>A short summary of the modelling approach</i> .....	15
3.3.3	<i>Implementation details</i> .....	16
3.4	RELEVANT RESULTS .....	19
<b>4</b>	<b>PLATOONING APPLICATION TEST-BED .....</b>	<b>20</b>
4.1	INTRODUCTION.....	20
4.2	USE CASE DESCRIPTION.....	20
4.2.1	<i>Dependability requirements</i> .....	20
4.2.2	<i>Scenario description</i> .....	21
4.3	IMPLEMENTATION DESCRIPTION .....	24
4.3.1	<i>Platooning application architecture</i> .....	25
4.3.2	<i>Decision making algorithm</i> .....	27
4.3.3	<i>Communication Aspects</i> .....	29
4.3.4	<i>Payload-Wormhole Interfacing Aspects</i> .....	29
4.4	ADDRESSING DEPENDABILITY REQUIREMENTS.....	30
4.5	SUMMARY OF RESULTS .....	32
<b>5</b>	<b>DISTRIBUTED BLACK-BOX TEST-BED.....</b>	<b>34</b>
5.1	INTRODUCTION.....	34
5.2	USE CASE AND DEPENDABILITY REQUIREMENTS .....	34
5.3	IMPLEMENTATION DESCRIPTION .....	35
5.4	SUMMARY OF PRELIMINARY RESULTS .....	39
<b>6</b>	<b>RESILIENT COMMUNICATION TEST-BED .....</b>	<b>40</b>
6.1	INTRODUCTION.....	40
6.2	TARGETED DEPENDABILITY MECHANISMS.....	40
6.3	RESILIENT COMMUNICATION TB SET-UP .....	41
6.3.1	<i>Scenario</i> .....	41
6.3.2	<i>Network architecture</i> .....	42
6.3.3	<i>Node Architecture</i> .....	42
6.3.4	<i>Topology emulation</i> .....	43
6.4	IMPLEMENTATION DETAILS.....	43
6.4.1	<i>Multi-Channel Multi-Radio</i> .....	43
6.4.1.1	Hardware implementation.....	43
6.4.1.2	Software components.....	44

6.4.1.3	Test/evaluation scenarios.....	44
6.4.2	<i>Fast Reroute</i> .....	46
6.4.2.1	Implementation.....	46
6.4.2.2	Evaluation.....	46
6.4.3	<i>Replication Manager</i> .....	47
6.4.3.1	Software components.....	47
6.4.3.2	Test/evaluation scenarios.....	47
6.4.3.3	Topology emulator.....	47
6.4.3.3.1	Use case in Resilient Communication test-bed.....	48
6.4.3.3.2	Deployment.....	49
6.5	SUMMARY OF RESULTS.....	50
<b>7</b>	<b>SUMMARY AND OUTLOOK.....</b>	<b>51</b>
<b>8</b>	<b>LIST OF ACRONYMS.....</b>	<b>52</b>
<b>9</b>	<b>REFERENCES.....</b>	<b>54</b>

# 1 Executive summary

## Overall objectives of WP6 & context of this deliverable:

This WP6 deliverable contributes to the HIDENETS main objectives, which are stated in the Technical Annex:

*HIDENETS addresses the provision of available and resilient distributed applications and mobile services with critical requirements on highly dynamic and possibly unreliable open communication infrastructure.*

These dependability solutions entail design approaches, communication/middleware functionalities and their corresponding algorithmic implementations, as well as analysis and testing methodologies. Hence, the main objective of WP6 is to

*Provide an implementation of the relevant parts of the design solutions to constitute a proof of concept prototype in the automotive application domain covering both ad-hoc car-to-car and server based (infrastructure) scenarios.*

The implementations therefore will demonstrate the feasibility of the HIDENETS approach and in addition will act as platform for the experimental quantitative analysis (WP4) and parts of the testing activities (WP5). Altogether these support the following HIDENETS goal:

*Perform an assessment of the dependability and QoS, provided by the solutions developed in HIDENETS, through the evaluation of the selected scenarios both at model resolution level and by experimentation on the proof-of-concept laboratory set-up.*

This deliverable contains the refined specification and implementation details of the proof-of-concept experimental set-up. The refined specification thereby is an extension of the previous D6.2, taking into account the architecture and resilience service modifications that occurred since then. During the second half of Year 3 of the HIDENETS project all these implementations will be evaluated and further documented in deliverable D6.4.

## Summary of the content of the deliverable:

This document contains the specification of the final version of the HIDENETS proof-of-concept laboratory set-ups, focusing on implementation details. These set-ups will be used to show the feasibility of selected parts of the HIDENETS solutions when implemented on COTS components, and to assess their impact on dependability and performance in an experimental laboratory environment. The laboratory set-ups are based on the application and use-case scenarios from D1.1 [1], the network and node architecture as specified in D2.1.2 [2] and D3.1.2 [3], as well as partially influenced by the work on quantitative analysis and design methodologies in WP4 [4] and WP5 [5]. This final specification of the HIDENETS proof-of-concept laboratory set-ups is based on the specification presented in D6.2 [6].

In order to focus on essential functionalities and to show the benefits of a flexible and modular HIDENETS architecture, four focus areas are identified which lead to the definition of four specialised test-beds:

- A test-bed showing a complex resilient application developed specifically on top of the HIDENETS solutions using model-based development methods. This application involves both the infrastructure and the ad-hoc domain.
- A Platooning application that is used as a proof-of-concept for the ability to detect and react to timing faults, to assure safety and to handle certain malicious intrusions.
- A Distributed Black-Box application showing the crucial middleware functionality that provides major dependability benefits.

- A test-bed focusing on resilient communication protocols for ad-hoc (car to car) networks and their impact on higher-layers.

These four test-beds are based on the same overall test-bed architecture. For each test-bed, selected parts of the physical components in the architecture as well as selected elements of the functional architecture are simplified. The latter is achieved by using stub implementations of the HIDENETS functional blocks that are not essential for the specific test-beds. The four test-beds both provide a stand-alone benefit and can subsequently be used to show additional benefits from other combinations of HIDENETS components.

Since the dynamicity of the communication link properties and of the topologies in the ad-hoc domain is one of the major characteristics of the HIDENETS scenarios, a topology emulation tool is developed and described in the context of the test-beds. This tool can replace actual wireless communication technology in all test-beds. The main motivation for the development and specification of this tool in WP6 is to allow for reproducible experiments with dynamic topologies.

## 2 Introduction

This document is a HIDENETS WP6 deliverable and contains the specification of the final version of the HIDENETS laboratory set-up and its components. This deliverable is an extension and refinement of the specification of the laboratory set-up as described in D6.2 [6] and provides a description of the implementation of application test-beds (see below) as well as summarized test-plans. The test-beds are composed of ad-hoc as well as infrastructure components and are used as proof-of-concept prototypes for HIDENETS results.

The aim of HIDENETS is to develop and analyze end-to-end resilient solutions for distributed applications and mobility-aware services featuring ubiquitous communication scenarios. These scenarios include ad-hoc domains and mobile networks connected to communication infrastructures. However, as communication links and common system components are inherently unreliable, end-to-end system-level resilience solutions must be developed, addressing both accidental and malicious faults. Analysis and validation of these solutions are performed in the context of the project via analytic/simulation models, and via experimental proof-of-concept set-ups (test-beds). A description of this HIDENETS proof-of-concept approach is the focus of this deliverable, which accompanies the actual implementations. Since the ambition of HIDENETS is to develop end-to-end dependability solutions, it is essential that a holistic end-to-end view is maintained throughout the project.

In order to achieve this goal within the timeline of the project, the solutions have been developed in parallel. Combined with the complementary nature of the HIDENETS solutions, this has been the motivation for the definition of four test-beds. Each of the four test-beds has its dedicated stand-alone goal, and in addition they also contain a selected part of the overall holistic HIDENETS solutions, as explained in the following:

- The Application Development (AD) test-bed validates selected mechanisms for resilient infrastructure-based applications, which is essential to cover the full set of HIDENETS application access scenarios. Furthermore, this test-bed validates the capabilities of the application development methodology to reduce the design and implementation faults on application level. In addition to this clear stand-alone benefit, this test-bed also has an integrating role. More advanced experimental scenarios include selected resilience services from the other test-beds, hence allow to validate the joint benefits in the newly developed application cases. Hence, this test-bed provides further evidence of the benefit of the HIDENETS holistic approach. In order to allow the integration of services from other test-beds, a process for definition and change control of service interfaces has been implemented across all HIDENETS work-packages.
- The Platooning (P) test-bed validates a set of mechanisms that allow to detect and to react to violations of timeliness requirements, and to malicious intrusions in widely required distributed consensus middleware functions. Causes of timing errors may be manifold, with communication delay and jitter that are especially pre-dominant in wireless multi-hop communication networks being a major class of faults. The use of additional HIDENETS services (from test-beds RC and P) will therefore lead to increased reliability and availability, as communication faults may be handled transparently at the lower layers. This does not lead to violation of timing requirements and consequently may reduce the probability of the system having to enter a safe state. This joint benefit will be quantified within the holistic model analysis of WP4.
- The Distributed Black-Box (DBB) test-bed validates the benefits from redundant, distributed data storage approaches, focusing mainly on crash-faults of the data-source. In addition, the impact of malicious, non-cooperative nodes is alleviated via the suitable cooperation oracles. The resulting reliability of the data storage (expressed by the probability of successful recovery of the data) would thereby be further increased, if multi-hop communication with the efficient and resilient mechanisms from HIDENETS (RC test-bed) are included.
- The Resilient Communication (RC) test-bed shows efficiency improvements due to resilient communication protocols in the ad-hoc domain, which are particularly relevant in resource-limited

scenarios in a highly dynamic environment. Therefore, the mechanisms that are validated via this test-bed are useful for fault-tolerance and even for avoidance of congestion faults that lead to timing or omission errors. Furthermore, these mechanisms allow the system to deal with link failure as well as crash faults of intermediate (relaying) nodes in a multi-hop communication path. Finally, via the implemented replication manager, resilience to crash-faults of service providing nodes can be achieved while minimizing the probability of application failures. As all these fault cases are handled on the communication layer and in lower middleware, the higher middleware functions and the applications will not be affected, given that the implemented recovery mechanisms are successful. Hence, the mechanisms in test-bed RC lead to a more 'stable' and reliable end-to-end message exchange as well as a more reliable application access in the ad-hoc domain. Such improved communication properties will be beneficial to all other test-beds.

In order to show the complementary features, **Table 2-1** describes the main design aspects of the individual test-beds. The rationale behind the prototyping approach relies on:

- Both the infrastructure and ad-hoc networking domains are covered by the four test-beds collectively, see first row of Table 2-1. In the ad-hoc domain, both single-hop as well as multi-hop communication is covered.
- The set-up of the Platooning and Distributed Black Box test-beds is driven by a specific application case. For the other two test-beds, the listed applications are mainly motivating examples (see Row 2 in Table 2-1).
- As the introduction of physical mobility for ad-hoc nodes leads to many implementation and reproducibility challenges, three of the test-beds rely on emulated node movements. To show the feasibility in real mobile scenarios, the Distributed Black-Box test-bed includes the physical mobility using moving, scaled-down vehicles, see Chapter 5.
- Each of the test-beds validates the improvements from the HIDENETS services for a certain subset of fault-types, see last row of Table 2-1. These subsets are partially overlapping and in total cover a wide range of faults as identified in the fault-analysis performed within WP2 and WP3.

Following this approach, the prototyping activities will show the feasibility as well as the achievable improvements by the holistic HIDENETS approach.

Together, these test-beds cover the major parts of the HIDENETS node architecture [2, 3], as shown in Figure 2-1. Note that this is a high-level view and that the ellipses are drawn to highlight the focus on specific HIDENETS functionality in each test-bed. Furthermore it should be stressed that these four test-beds represent a collection of integrated components, and together cover all relevant HIDENETS aspects.

A description on the set-up and implementation of each test-bed is provided in chapters 3 to 6. The deliverable is concluded with a summary and outlook in chapter 7.

**Table 2-1 Overview of the main design aspects of the individual test-beds**

	<b>AD test-bed</b>	<b>P test-bed</b>	<b>DBB test-bed</b>	<b>RC test-bed</b>
<b>Network Scenario</b>	Ad-hoc + Infrastructure	Ad-hoc	Ad-hoc + Infrastructure	Multi-hop Ad-hoc
<b>Application</b>	Platoon Driver Support Software (PDSS).	Platooning	Distributed Black box	Simplified versions of video streaming and blackboard
<b>Node mobility</b>	Static and emulated mobility	Emulated mobility	Physical mobility	Static and emulated mobility
<b>Wireless links</b>	WLAN	802.11g, Optionally emulated	802.11g	802.11a, or emulated
<b># nodes</b>	≥ 4 ad-hoc 2 infrastructure	4-7 ad-hoc	≥ 4 ad-hoc 1 infrastructure	4-7 ad-hoc
<b>Ad-hoc node structure</b>	Split: embedded devices	Split: payload + wormhole part (sensor)	Standard single node	Split: processing unit <-> multi-interface HW
<b>Implemented HIDENETS services</b>	HIDENETS/SAF services (stub/implemented) messaging, check pointing, "secure" channels, authentication, freshness detection, Reliable and self-aware clock	Authentication; Timely Timing Failure Detection; Duration Measurement; Reliable and self-aware clock; Intrusion tolerant consensus; QoS Coverage	Cooperative backup; Trust+ Cooperation oracle; Proximity map	Multi-channel / multi-radio management; Fast Re-Routing; Replication manager
<b>Goal: fault categories</b>	Application design & implementation, infrastructure node failure	Malicious "intrusion"; Timeliness; Safety	Malicious, non-coop. behaviour; Node failures "crash of source"	Congestion; Link breaks; Node failures (relay+end nodes)

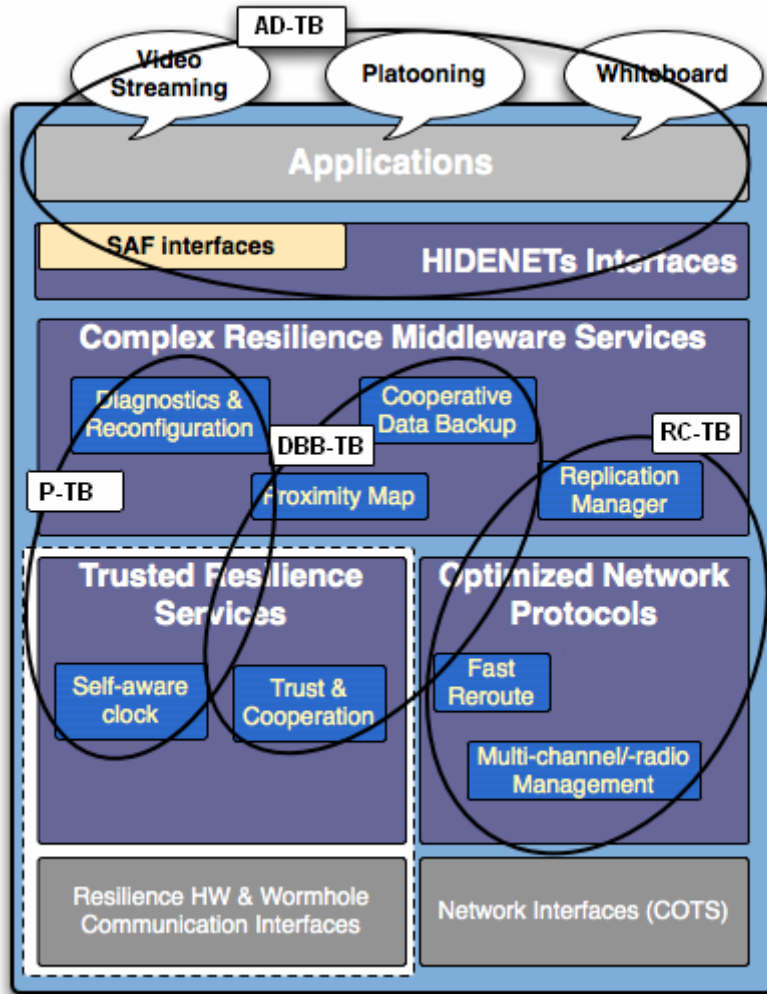


Figure 2-1: The scope of the test-beds with respect to the HIDENETS node architecture.

### 3 Application Development test-bed

This chapter presents a brief overview on the Application Development (AD) test-bed (TB), an SAF AIS based test-bed of the HIDENETS project. Section 3.1 enumerates key goals of the AD TB. Then section 3.2 outlines the actual application used for demonstration. The HW/SW requirements and implementation details are discussed in section 3.2.2, while achieving high availability and some details related to the modelling effort are discussed in sections 3.3.1.2 and 3.3.2 respectively. Section 3.4 highlights the most relevant results achieved by the AD TB.

#### 3.1 Introduction

The AD TB uses an *example application* in order to demonstrate (i) the benefits of applying the *model driven software engineering methodology* and tools developed in the context of WP5 and (ii) the possibilities for achieving *high availability* of some key infrastructural services by building them on a standards compliant SAF AIS implementation:

- The *example application* developed for the AD TB is a platoon driver support software (PDSS) application. PDSS is a distributed, partially embedded system consisting of (i) *embedded components* playing the role of sensors and actuators, (ii) some components deployed to nodes in the *ad-hoc domain* and (iii) some highly available nodes in the *infrastructure domain*. Note that the example application of the AD TB primarily serves demonstration purposes: with a focus on a restricted subset of functionalities building up a real-world application. This subset was chosen to demonstrate the feasibility and usefulness of some core services in the HIDENETS architecture in a real world application environment. Furthermore, although PDSS is based on a subset of services developed in the context of the HIDENETS project, it is not meant to be a hard real-time, safety-critical application.
- The *model-driven design methodology* developed in the context of WP5 was applied during the design of the PDSS application. The goal of the AD TB in the context of PDSS is to demonstrate, how to exploit the modelling facilities delivered by WP5 including the well-established HIDENETS meta-model, UML profile for HIDENETS and the corresponding design patterns during the design and implementation of an application built on the HIDENETS platform<sup>1</sup>.
- The *high availability* of PDSS services in the infrastructure domain was achieved by building them on an SAF AIS implementation. In this context, the AD TB demonstrates the benefits of using the SAF-related meta-model, profile and design patterns developed in WP5. Another key focus of the AD TB is the demonstration of automatic configuration synthesis tools developed in WP6; these solutions were used for the automatic construction of configuration files for SAF AIS middleware. The automatic configuration file generation has been proven to substitute the tedious and error-prone manual configuration with a straightforward synthesis process directly based on the model of the application.

#### 3.2 Requirements considered for the test-bed

When designing our test-bed system we had to consider two fundamentally different sets of requirements. The first set of requirements is determined by the application field (cooperation of vehicles, fleet management, ...) and we refer to them as “use-case requirements”. The second set is determined by the underlying hardware and software components (enterprise hardware, personal equipment, devices with low computational power, HA and communication middleware, ...) that are considered already given, and we refer to them as “HW/SW requirements”.

##### 3.2.1 Use-case requirements

This section outlines the example application used in the AD TB. The developed software is a simplified version of an intelligent cruise control application (mentioned in the context of the “platooning use-case” in

---

<sup>1</sup> Independent of its implementation in hardware or software

previous deliverables [1, 6]); it will be referred to as the Platoon Driver Support Software (PDSS). Below we briefly define the key concepts of PDSS and enumerate the functional and non-functional requirements to be satisfied by the application.

The platoon in the framework of the PDSS application consists of vehicles, which are driven by human drivers. The first vehicle in the platoon is called the *head vehicle*; other vehicles in the platoon are called *slave vehicles*. Slave vehicles should adjust their speed as dictated by the head vehicle. This speed adjustment is supported by the PDSS software. The *functional requirements* of the PDSS application can be summarized as follows:

- The software periodically *collects various parameters* of vehicles (speed, acceleration, etc.) and, using the head vehicle as reference, calculates the actuations to be applied to slave vehicles.
- The software presents the *map of the current area* to the driver of the head vehicle similarly to a usual GPS device with additional annotations indicating various traffic circumstances; the database of traffic conditions is stored in the infrastructure and managed by (human) operators.
- The software periodically reports the *actual position of the platoon* to the infrastructure. On the infrastructure side the operators can follow the movement of platoons on a display.

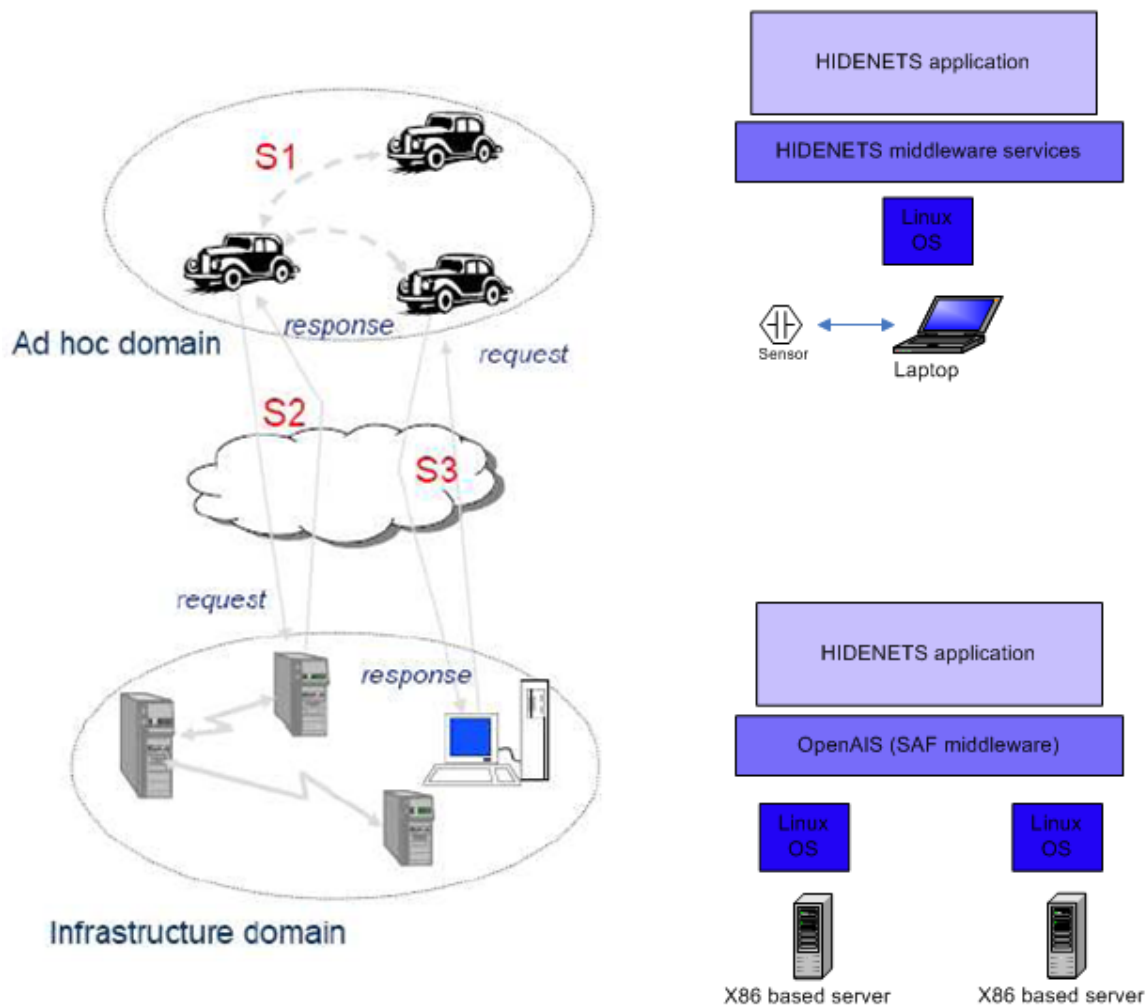
The *non-functional requirements* of PDSS are as follows:

- Controlling slave vehicles requires *up-to-date information* about the actual state of the vehicles and the actuations calculated by the software should be applied to the slaves within short deadlines.
- The data channel between vehicles should be *tamper proof*, i.e., communication should be encrypted, and peers should be authenticated.
- The decisions of platoon drivers are taken according to up-to-date traffic conditions. Thus the information provided by the infrastructure should be *trusted* and peers of this communication should be *authenticated*.
- The infrastructure side provider of traffic information and the infrastructure side database of actual platoon positions should be *highly available*.

It is easy to see that the first two non-functional requirements are supported by the HIDENETS services, Timely Timing Failure Detection and Authentication respectively (we used a stub implementation of these services,) while the availability of infrastructure services was achieved by building them on an SAF AIS middleware using the “2N plus spare” redundancy model (i.e., if a component becomes unavailable, another component takes over the service with the assistance of the failover facility). This organization of infrastructure services ensures that no data transaction shall be lost because of hardware or component failures.

### 3.2.2 HW/SW requirements

The application is implemented in a diverse environment consisting of both *enterprise hardware* (for the server components in the infrastructure domain), *personal equipment* (for the components in the ad-hoc domain) and *devices with low computational power* (for the sensors). The *enterprise environment* consists of the following general purpose (i.e., not HIDENETS specific) HW/SW components: (i) a *cluster* of IA32 [10] based nodes running Linux, (ii) SA Forum compliant middleware (OpenAIS [14]) in the infrastructure domain, responsible for node and application monitoring, node exclusion in failure situation and split-brain avoidance. The centre of intelligence built into a vehicle (*personal environment*) is a laptop running Linux and a simplified version of the HIDENETS middleware i.e., some services are substituted by stubs. The sensor and actuator components are emulated by embedded devices built on Atmel microcontrollers.



**Figure 3-1 Overview of the experimental set-up**

There is no preference in the nature of the communication and networking part, as it is expected that the delivered resilient architecture and Hidenets communication services will hide their specific details. A general architecture for the test-bed, consisting of a cluster in the infrastructure domain plus a set of nodes in the ad-hoc domain, is shown in Figure 3-1. The nodes in the ad-hoc domain run as separate processes on the same laptop, using RPC calls to emulate the environment.

### 3.3 Technical details of the test-bed

After considering the above mentioned requirements we had to design the application development test-bed. This section focuses on some technical key points of the test-bed: the applied middleware systems, the applied application development approach and some implementation details.

#### 3.3.1 Applied middleware components

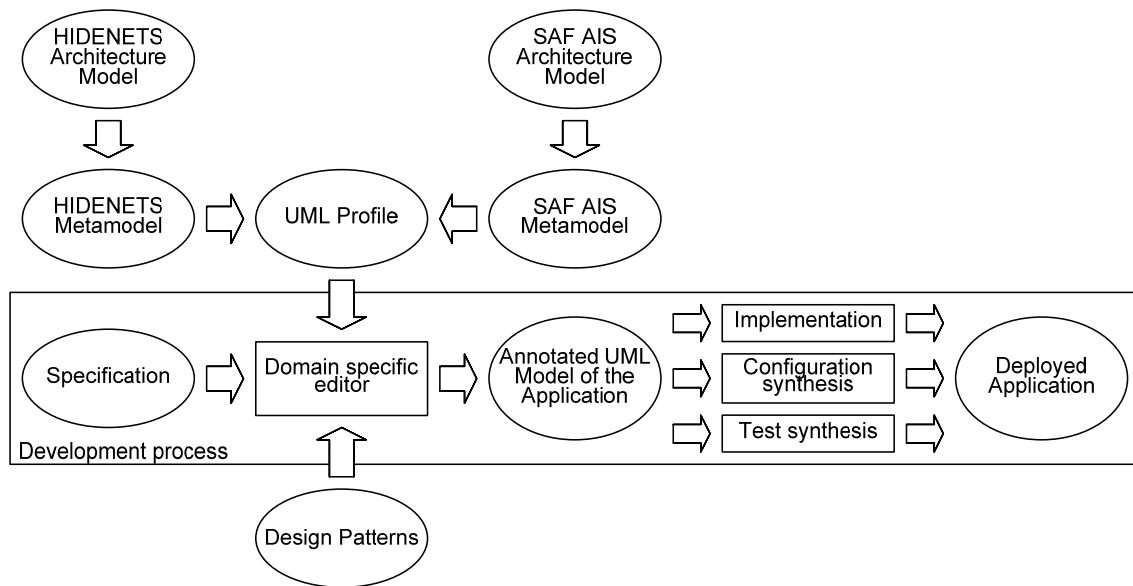
The application development test-bed demonstrates how the usage of the Hidenets middleware services can simplify the application development. When considering the Hidenets based systems we distinguish between system components running in the ad-hoc domain and the other ones running in the infrastructure domain. The difference in the application environment has consequences in both the requirements for the components and the resources that can be expected. Therefore we have chosen different middleware systems for the two domains: the Hidenets middleware for the ad-hoc domain and a SAF-AIS compliant middleware for the infrastructure domain. This subsection describes the main attributes of these two middleware systems.

### 3.3.1.1 Application of the Hidenets middleware

The modules of the ad-hoc domain of our implementation rely on the Hidenets middleware that was developed in WP2 and WP3 and is described in detail in [21, 22, 23]. Our implementation utilizes services of the wormhole-kernel, like TimelyTimingFailureDetection, AuthenticationService and Reliable&Self-AwareClock -- described in [17] -- focusing on timeliness and authentication oracles thus keeping the functionality as simple as possible. These services are replaced by stubs that provide a simplified functionality yet mimic the interfaces made available for applications.

### 3.3.1.2 Application of a SAF-AIS Compliant Middleware

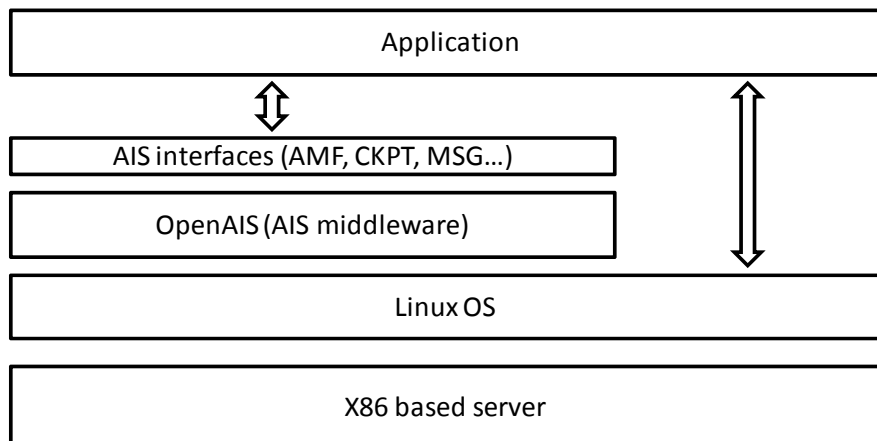
This section enumerates the key features of the SAF AIS middleware used for the implementation of highly available infrastructure services. Moreover, we summarize the steps of our modelling efforts the benefits of which were demonstrated by the development of the AD TB. The relationship between modelling, synthesis and testing efforts is shown in Figure 3-2.



**Figure 3-2 Relationship between modelling, synthesis and testing efforts**

### 3.3.1.3 A short summary of the High-Availability (HA) oriented approach

The discussion below (see also Figure 3-3) briefly summarizes those SAF AIS features that play an important role in the achievement of infrastructure services' high availability.



**Figure 3-3 Cluster node in the AD test-bed**

The Availability Management Framework (AMF) takes care of monitoring HA applications and failing them over if needed (i.e., restarting the application in case of failure). Redundant instances of the application run on the same node or on different nodes. The AMF monitors the correct functioning of the application, reports errors and initiates recovery actions if needed. The following fault cases are handled:

- Failure of a HW or SW resource represented by an AMF component. The failure is detected and handled according to configurable policies.
- Failure of an entire node. If configured, services from the failed node are taken over by standby nodes.

The additional SAF services in the AD TB provide (i) highly available communication resources, (ii) replicated checkpoints, (iii) cluster-wide application locks and (iv) membership service.

A *highly available communication resource* (SAF message queues, SAF events) is typically owned by the active instance of an application. When the active application fails, the ownership of the HA communication resource is transferred to the standby application which is taking over the active role. The communication resources provided by the SAF Message Service are (i) message queues, (ii) message queue groups, and (iii) event channels:

1. *Message queues* allow for a multipoint-to-point communication between senders and receivers. There may only be a single receiver on a queue at a time, but the lifecycle of a message queue is decoupled from the lifecycle of its receiver. A sending process puts messages into a queue without being aware where the queue is physically located. Messages are versioned and time-stamped by the sender.
2. An *event channel* is an object to which event publishers and event subscribers may attach themselves in order to exchange events. An event consists of a standard set of event attributes and the event data. An event is time-stamped by its publisher. Event channels allow for a multipoint-to-multipoint communication.

An active application preserves its current state information in a *checkpoint*. Checkpoints represent especially the state of ongoing service requests. Checkpoints are *replicated* between nodes so that the information is not lost if a node fails entirely. When an active application fails, the corresponding application synchronizes itself according to the preserved state information and can seamlessly resume the activity of its broken counterpart.

If an application needs to ensure that actions with a cluster-wide scope do not interfere with simultaneous actions of another application, it can use *locks on associated cluster-wide resources* to synchronize the parallel activities.

Although this service is available for use by applications, it is usually preferred that applications exist within a single system image of the cluster rather than being aware of the cluster configuration and the states of all nodes. The *cluster membership service* is primarily used by the AMF and the SAF service to update the states of their maintained cluster-wide resources (components of HA applications, message queues, checkpoints, etc.) according to which nodes are currently members in the cluster.

### 3.3.2 A short summary of the modelling approach

The ultimate goal of the HIDENETS project is to provide a set of communication and middleware services to support the realisation of distributed applications and mobility-aware services in ubiquitous communication scenarios. The results can only gain interest if support for the application development can be provided. WP5 aims at the integration of the HIDENETS concept with state-of-the-art SW engineering methods covering modelling, development and testing phases (further details will be provided in the upcoming deliverable D5.3). The modelling efforts, the benefits of which are demonstrated in the AD TB, are organized into a multi step process (see Figure 3-2):

- First the *UML models of the involved platforms* (HIDENETS and SAF AIS) were constructed. The structure of implementation platforms was captured by static structure models (i.e., class and component diagrams), provided calling APIs were introduced into the model by the definition of

interfaces while the intended usage of services was shown on interaction models. This phase answers the question: “What are the interfaces of services exposed by the platforms to application developers, and how are these interfaces used within the platforms for interconnecting services within the frameworks?”

- On top of the basis of the development platforms’ models WP5 constructed platform (Hidenets and SAF) specific *meta-models* of those application parts that are aimed to be integrated with one of them. These meta-models illustrate the intended organization of applications running on the platforms and identify the key concepts from an application developer’s point of view with respect to various services, introducing the corresponding meta-classes and connecting these newly introduced meta-classes to core UML concepts. This step answers the question: “What is the intended organization of applications running on the platforms and how do the platform-related parts correspond to fundamental UML concepts?”
- Having constructed the meta-models, a *UML profile* was constructed on the basis of these meta-models; the profile defines stereotypes and tagged values to be used for annotating UML models with dependability and platform-related information. This step answers the question: “How to add development platform related information to ordinary UML models of dependable applications that are intended to take advantage of the platform capabilities?”
- Finally we provided a set of *design patterns* to support the implementation of applications built for the development platforms using our profile. These patterns can be seen as detailed examples for implementing various dependability-related parts of applications. This step answers the question: “What are the best practices for organization and implementation of various dependability-related parts of applications that are intended to be executed on the HIDENETS/SAF AIS platform?”

Having constructed the UML profile and defined the design patterns, the compilation of the *domain specific editor* (DSE) was quite straightforward: we saved the UML profile in a format compatible with the IBM Rational Software Architect modelling environment, and next we extended the profile’s features by various graphical notations that provide easy to recognize visual notations for the modeller.

### 3.3.3 Implementation details

As described above, the AD TB followed the Model-Driven Development (MDD) process to construct its application (PDSS). The use-case driven nature of MDD resulted in a functionally decomposed design, depicted in Figure 3-4 showing an excerpt of the PDSS model (ad-hoc side). There are basically two types of actors: the SlaveVehicle(s) and the HeadVehicle with their various modules being responsible for actuation or communication with the fixed infrastructure.

On the other hand the class diagram shown in Figure 3-5 presents a fragment of the PDSS infrastructure side’s static model where PDSS services are built on such SAF concepts as service groups or service units. The diagram depicts the software structure of the application (top-level components) and outlines the deployment of key components to computing resources. The highlighted part of PDSS consists of a single service group implemented in a redundant scheme and supported with health check functionality. The SAF AIS cluster is formed on a standalone server with the nodes running on virtual machines.

The individual ad-hoc components are implemented in standard C language and they communicate through Remote Procedure Calls (RPC). Both actors run on the same laptop, and the RPC is used to emulate the unpredictable ad-hoc environment.

The services in the infrastructure part are also implemented in standard C, using the SA Forum AIS interfaces for availability management, state preservation and communication. The communication between the infrastructure and the ad-hoc part of the application is carried out through standard HTTP protocol over TCP/IP connections. In order to avoid overhead in the ad-hoc components, IP failover is realized in the “gateway” component to avoid the occurrence of single points of failure in the architecture. This virtually means that the service is accessible always on the same IP address and the ad-hoc side clients can use that for communication.

---

The in-cluster communication on the infrastructure side is based on the SAF Message service. As mentioned in section 3.3.1.3, this service provides simple multipoint-to-point communication scheme which fits our services much better than the publish-subscribe scheme which is provided by the Event service. In addition, the events have a determined retention time after which they are removed from the system, but before removal, all new subscribers get the message. In our case all service provision is done on the FIFO scheme which does not require such distribution of data and the messages should be delivered to only one recipient and exactly once.

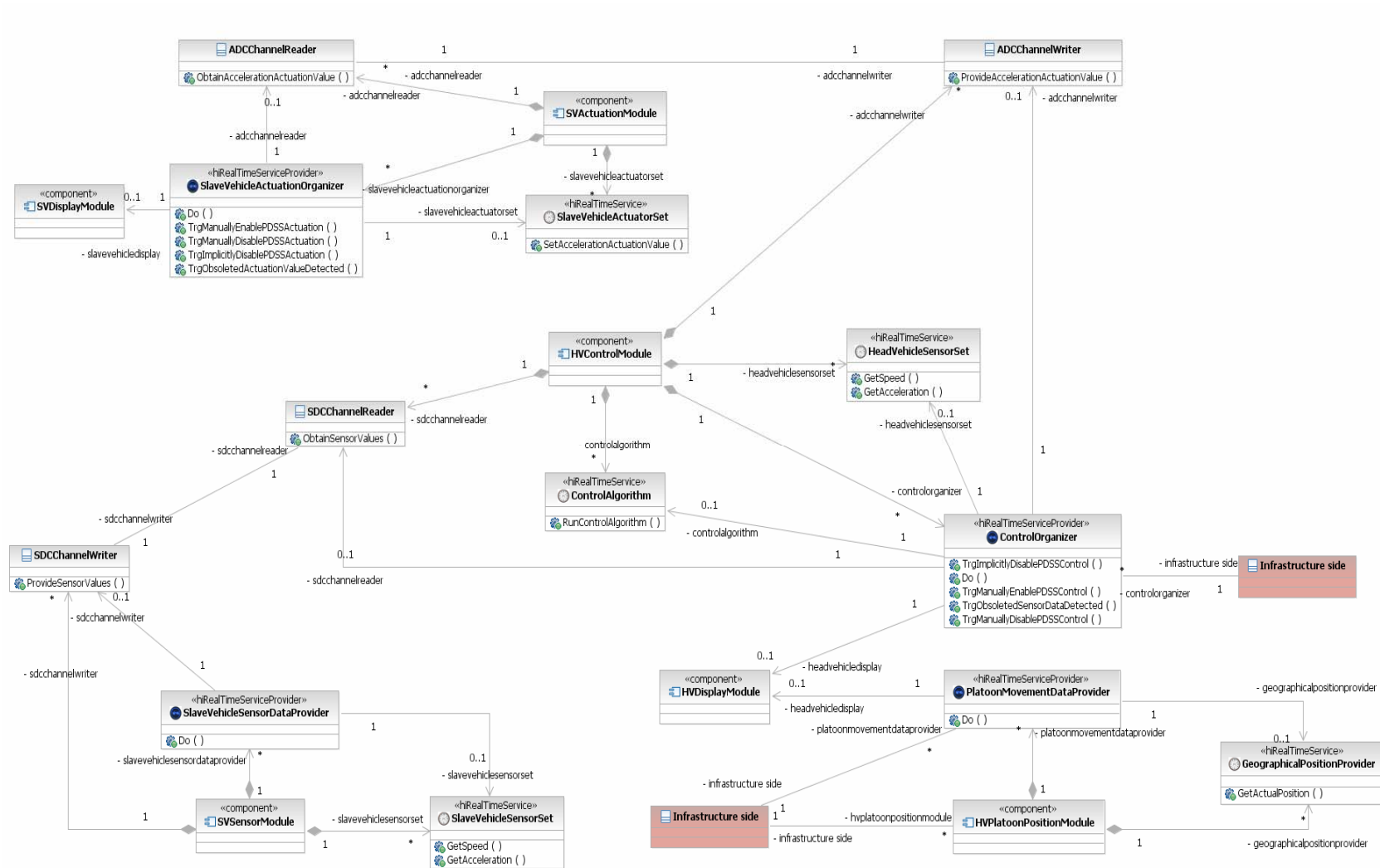
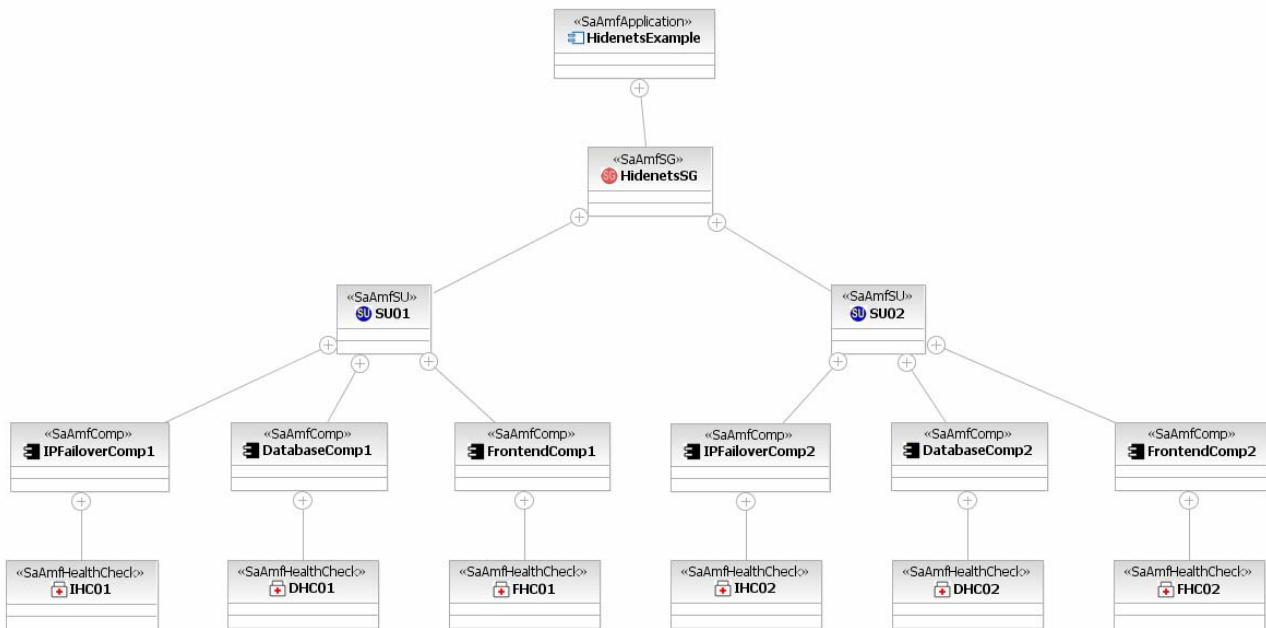


Figure 3-4 PDSS model (ad-hoc side) – excerpt



**Figure 3-5 PDSS model (infrastructure side) – excerpt**

### 3.4 Relevant results

The AD TB has successfully demonstrated the viability of efforts carried out in WP5 concerning the model-driven development. For WP6 the AD TB has provided results regarding the automatic synthesis of complex configuration files directly from the model of the application. Furthermore our experiences with the application of SAF AIS middleware provided valuable feedback to the SAF community:

- The UML model of the PDSS application was designed using the *UML profile* developed in WP5: software components running on ad-hoc nodes were annotated by HIDENETS platform related stereotypes, infrastructure components were annotated by SAF AIS related stereotypes and the implementation of this platform specific model was carried out by the systematic application of design patterns.
- The *configuration files* for the SAF AIS middleware were *automatically synthesized* from the model of the infrastructure side's software using our automatic configuration file generation solutions developed in WP5.
- PDSS's *infrastructure services* were implemented on top of SAF AIS middleware; the high availability of vital services was ensured by the redundancy model and SAF AIS's automatic fail-over capabilities.

## 4 Platooning application test-bed

### 4.1 Introduction

The Platooning (P) application test-bed (TB) constitutes a proof-of-concept set-up for a subset of the concepts and services developed within WP2 and WP3 (namely Task 3.3). It is concerned with communication aspects by considering a distributed application of which the dependability characteristics are strongly dependent on the communication system properties. In fact, the idea of this test-bed is to illustrate the dependability, resilience and performance benefits that may be achieved for the considered application, with the help of some selected wormhole services and also by using some complex services from the higher middleware layer.

The specific services addressed within this test-bed are the following: Reliable and Self-Aware Clock, Duration Measurement, Timely Timing Failure Detection, Authentication, QoS Coverage Service, Diagnostic and Reconfiguration Manager and Intrusion Tolerant Agreement. Therefore, this test-bed complements the other test-beds described in this deliverable.

The selected application for this test-bed was the **Platooning application**. In fact, in HIDENETS we have identified some applications that could be considered as potentially interesting to demonstrate the results of the project. In particular, we can mention some of the applications that are included in the Assisted Transportation use case, such as the Floating Car Data (FCD) application, the Hazards Warnings and Information from other vehicles application (HzW), the Maintenance/Software Updates application (MSU), and also the Platooning application and use case (see [1]).

However, among the several possibilities the Platooning application is particularly attractive because it involves safety critical aspects, which are interdependent of timeliness aspects and of the assurance of temporal consistency guarantees. In fact, the dissemination of information and coordination among platooning members must satisfy some safety conditions (e.g. be timely, accurate, or else allow timely failure detection and reaction) in order to ensure that cars will never collide (at least, not because of the Platooning application). In addition, the Platooning application also presents significant trustworthiness requirements, making it very suitable for the purposes of this test-bed since all these requirements may be addressed with the solutions and services mentioned above, which were developed in HIDENETS.

In the next section we describe the Platooning application, with a focus on the dependability requirements and an overview of the scenario considered for the proof-of-concept set-up.

### 4.2 Use case description

#### 4.2.1 Dependability requirements

The objective of this application is to provide driving assistance for coordinated manoeuvring of vehicles. In general, the objective is to have all the cars closely following each other, at constant speed. To accomplish this, platoon members exchange sensor data and other information (positioning, heading, speed, acceleration), which is processed and used by the Platooning application to control the speed of the vehicle, while it can also be presented to the driver through a convenient interface. The driver may be able to take independent action, taking over the control of the vehicle despite the existence of automated car control functions.

The Platooning application involves a set of participant cars, which have to communicate through wireless channels in an ad-hoc environment. It is possible to consider different roles of the participants, namely a leader car and follower cars. So it may be possible to create scenarios in which cars have to *cooperate in a peer-to-peer fashion* (with information exchange among all the cars, possibly using broadcast modes of communication), as well as scenarios in which *direct communication* between the leader and one or more followers is envisioned.

Different safety requirements may be considered for the correct operation of this application. For instance, a non safety-critical requirement is that all the cars in the platoon *remain connected with each other* in the ad-hoc network, and a safety-critical requirement is that *car collisions never occur*. These non-functional requirements can be translated into requirements meaningful for the application, and expressed in terms of maximum and minimum distances that have to be considered as thresholds for taking appropriate action. For

instance, to ensure connectivity, one can state that cars *must remain close enough* (expressed by a maximum allowed distance). On the other hand, to avoid collisions cars *must remain far enough* from other cars (expressed by a minimum distance), to encompass for inaccuracies of available data at the application level as well as the necessary reaction time for braking a car until complete stop. We believe that it is possible to satisfy these safety requirements despite the ad-hoc nature of the communication environment (which implies improving dependability attributes) with the help of the envisioned architecture and services.

The Platooning application has intrinsic adaptation capabilities that may be explored in order to better deal with the uncertain characteristics of the considered ad-hoc environment. For instance, it is possible to control and adapt some communication parameters (e.g., the communication period, if periodic data exchange is used, the communication scope, or the redundancy level), or even changing the control algorithms depending on the operational context. Taking the Platooning application in more general terms, not just as a stand-alone piece of software, but as a piece of software that is used by “car objects” that altogether constitute the Platooning application, it is interesting to observe that each of these “car objects” admits at least one fail-safe state, which is stopping the car if some problem or failure is detected that could compromise the fail-safe requirement. Obviously this is only true if the car can be stopped in a way that does not put the other cars in danger (which should be true if the same reasoning is applied in all the cars). Nevertheless, other approaches may be considered fail-safe as well, depending on the specific failure. If the cars are just in risk of losing contact, then a possible fail-safe approach is to make every car agree on “meeting point” to which they should converge, so that they are always able to reconnect. The possibility of defining fail-safe states is interesting as a “last resort”, yet dependable solution, to deal with possible violations of safety properties. One important aspect that must be mentioned is that the implementation of fail-safety measures must be handled with extreme care in order to guarantee that the component responsible for the detection of the failures and triggering of the appropriate action is itself highly dependable and trustworthy.

It is possible to define **timeliness requirements** to be met by the Platooning application. In fact, for the application to be useful, it will need to incorporate decision making algorithms to control the speed of the car or just to help the driver within the platoon. However, for the correct operation of these algorithms, it will be necessary to have sufficient and accurate information about the real environment, that is, it is necessary to have dependable context awareness. Given the physical environment in which the application is used, reflected in the context information, there are timeliness requirements that must be met or, at least, timeliness information of which the application should be aware, in order to create a dependable context information repository. Some mechanisms to deal with these timeliness requirements are studied and addressed in HIDENETS, and can thus be demonstrated through this proof-of-concept set-up.

Finally, the Platooning application also exhibits **security requirements**, in the sense that it is necessary to have sufficient trust on the exchanged information in order to have valid outputs from the decision making algorithms. These kinds of requirements can also be addressed with the help of HIDENETS solutions, in particular with the help of an authentication service.

#### 4.2.2 Scenario description

The Platooning application is intended to provide speed control of vehicles so that these will operate safely as a platoon on a highway. In order to offer some flexibility, the driver is allowed to control the speed of the vehicle, with priority over the application. This means that the ultimate control of the car can be considered to be in the possession of the driver. The platoon will be formed by a group of vehicles that must always remain close together until they arrive, altogether, at their destination. In the considered scenario we assume that:

- vehicles always move in an approximately **straight line**
- there are **no other obstacles except other vehicles** moving ahead or behind
- cars always move **in the same and common** direction
- a platoon is formed by **two or more vehicles** following each other closely
- the cars forming a platoon will execute **coordinated activities**, namely to decide a common platoon speed

- the decision making algorithm in each car may temporarily determine a **different local speed** as required to maintain the car within a convenient distance from the preceding car
- in a steady state, however, the decision making algorithm will set the car with the decided global speed.

In general, platooning requires car-to-car communication and may include car-to-infrastructure communication. In the envisioned scenario, however, we will focus on **car-to-car communication**, as shown in Figure 4-1. Transmitted data includes the information about the position and instantaneous speed of a vehicle (this may be useful for the design of more complex but intelligent control algorithms), as well as a timestamp and, obviously, an identity of the vehicle. This data is transmitted periodically.

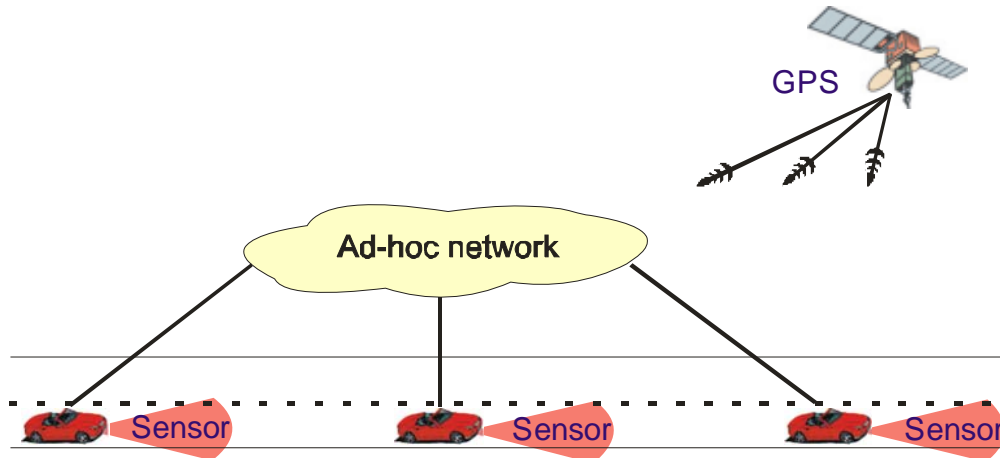


Figure 4-1: Communicating vehicles in a platoon.

Based on positioning information, longitudinal control of the vehicle is provided in order to maintain the distance to adjacent vehicles within the platoon (similar to adaptive cruise control). We do not consider lateral control via automated steering, given that we assume lane-keeping to be ensured by default and we do not deal with vehicle manoeuvres. Positioning information of each vehicle is obtained from a **GPS subsystem**, which we assume to be available (some specific interfacing aspects between the different subsystems are addressed in Section 4.3.4).

The fundamental requirements are expressed as the **minimum distance** ( $\Delta_{\min}$ ) to the front vehicle and the **maximum distance** ( $\Delta_{\max}$ ) to the follower vehicle, as shown in Figure 4-2. These requirements are established for each vehicle, depending on locally defined values for some crucial variables. The former ensures that, given the value for the maximum vehicle speed, an upper bound for the control loop interval and a maximum braking distance for full stop (or simply maximum deceleration), a vehicle will never collide with the preceding one. The latter ensures that, given similar bounds, a vehicle will not get too far apart from the follower one, as a means to ensure network connectivity. It also requires the knowledge of specific network parameters with influence on the wireless communication range.

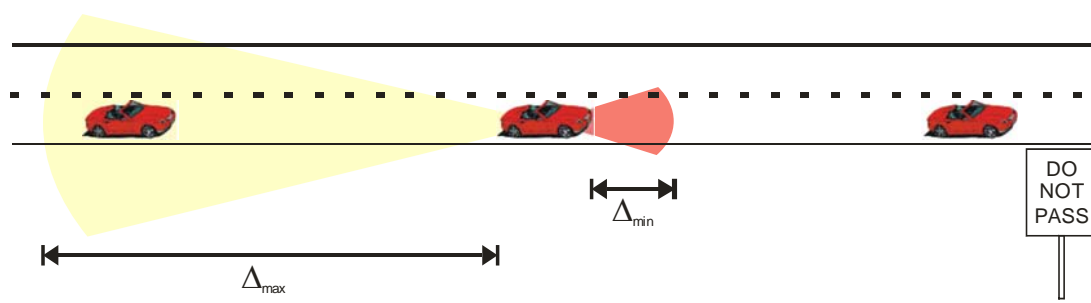


Figure 4-2: Fundamental requirements: safety distances.

Very importantly, we assume that vehicles are equipped with a **proximity sensor** that may be used to determine the actual distance to the preceding vehicle. In fact, we assume that the car is equipped with an internal safety system, which uses this distance information and which is able to directly actuate on the braking subsystem in order to stop the car in critical situations. This safety system is designed as a special purpose real-time system and is thus subject to the limitations imposed to such a predictable system. In particular, the information made available to this system is rather limited (we consider that it only knows the distance information) and all other variables (like the speed of the vehicle in front, for instance) are unknown and must be assumed in a pessimistic way (the front car must be assumed to be stopped, for instance). Because of this pessimistic approach, a car would be prevented to keep moving close to the preceding car without being forced to stop by the internal safety system.

Differently from the internal safety system, the Platooning application is executed in a separate environment, where it can collect additional context information and may thus be able to run a more optimistic decision making algorithm. In this sense, it would be preferable to control the speed of the car using the Platooning application, which can also request the car to stop in case some of the previously mentioned safety distances are compromised. The problem is that the execution environment of the Platooning application is not predictable and thus it is difficult to ensure that control decisions are made on time, as necessary to secure the main safety-critical requirement (collision avoidance).

In order to overcome the limitations of the internal safety system (reduced available information) and of the Platooning application (uncertain timeliness), we consider an application scenario in which both subsystems will play an important role in a hybrid manner: the internal car safety system will be used only when the Platooning application is not able to execute with the required timeliness, and it will be used as soon as this lack of timeliness is detected. The timeliness of the Platooning applications will necessarily have to be monitored by some timeliness car subsystem, which is able to switch the safety control to the internal car safety system when lack of timeliness is detected.

Further to what has been mentioned above, we assume that there is a gateway between the application execution environment and the internal car environment, which provides an interface to allow interactions between the two environments. In particular, it is through this gateway that the Platooning application can obtain information from the internal car sensors, and also send speed control information (braking and acceleration commands).

In the considered scenario we assume that public keys of every other vehicle have been previously uploaded to each vehicle.

The scenario will serve to demonstrate, in general:

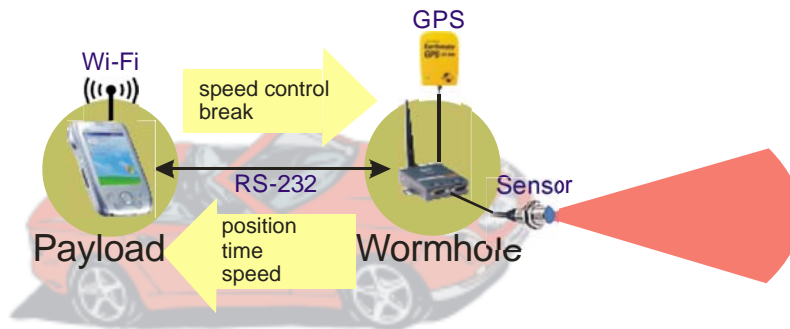
- The ability of the application to deal with unexpected changes in the values of relevant variables, such as: speed of vehicles, communication delay, distance to preceding or following vehicles. This will be done by means of adaptation, either as a normal and immediate result of a decision making logic, sending corrective indications to the relevant actuators within secure time limits or varying the decision intervals, or done as a self-reconfiguration procedure of the control logic by adjusting some of its parameters (e.g., tuning for higher speed, with more aggressive timeouts and consequently more timing failures of the application and a less smoother driving experience, or tuning for slower speed with less aggressive timeouts and less failures and thus a more pleasant driving experience).
- The ability of the application to improve the car behaviour with respect to securing the fundamental safety-critical requirement (avoid a collision with the preceding car). This will be done by integrating the decision making logic of the Platooning application with the fail-safety procedures defined for the vehicle.
- The ability to resist to intentional attacks to the integrity of information, which will be done by securing the critical information elements.

### 4.3 Implementation description

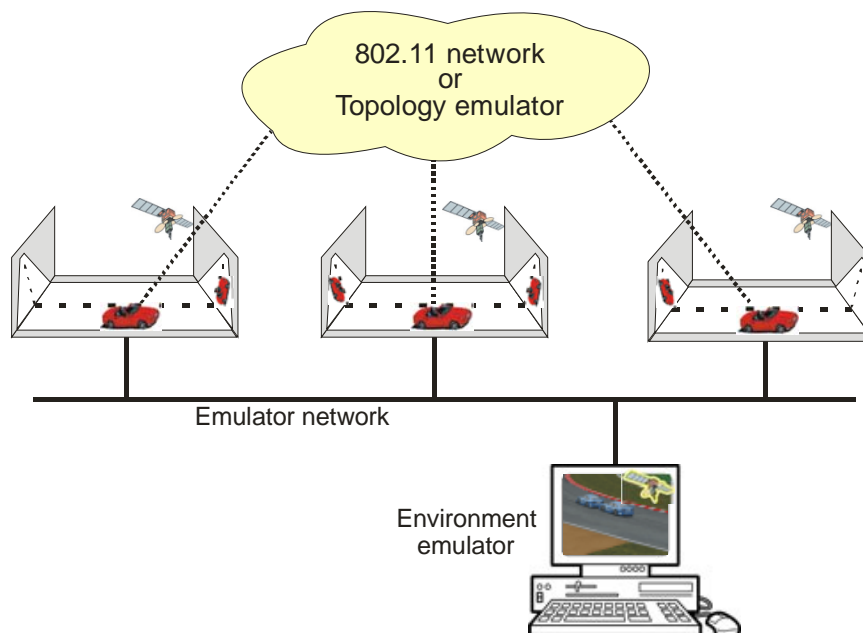
The Platooning test-bed implements the application scenario described in the previous section. Each car of the platoon (a node) is constituted by a computational device (a PDA or a laptop) that represents the payload (where the Platooning application is executed), and by an additional device with an embedded processor, which is considered to be part of the car infrastructure and takes the role of a wormhole (see Figure 4-3). In the current implementation the nodes are effectively deployed using laptops and for the embedded devices we use Lantronix UDS100 devices. This is in accordance with the architecture defined in HIDENETS, which is described in [2]. Each laptop is connected to a UDS100 device by a serial RS232 interface, through which the latter (the wormhole device) can provide its services to middleware components and applications residing in the payload device, in particular to the components that are used by the Platooning application.

On the other hand, other physical devices, like antennas (e.g. for GPS), sensors (e.g., for proximity detection) and actuators (e.g., for speed control), are considered to be part of the car infrastructure, similarly to the wormhole, to which they are connected in a predictable way. Therefore, all the relevant information for the Platooning application that concerns these physical devices (received from, or sent to them) will flow through the RS232 serial line and through the wormhole interface.

The Platooning application, residing on the payload system, processes different kinds of data: data received from the internal car devices (through the wormhole) and data received from other cars, by means of the communication in the ad-hoc network.



**Figure 4-3: Components of a car.**



**Figure 4-4: Emulated platoon.**

The “car” that we described above is not a real car, and will not be running on real roads. In fact, in the proof-of-concept set-up all of the mentioned hardware devices are **emulated**, as well as the physical world (or the environment) on which the “cars” of the Platooning application are instantiated. This means that we do not use real GPS devices, but emulated ones, and the proximity sensor is, once again, an emulated device.

This emulation environment is represented in Figure 4-4. The used emulator is available as free software, and is specially designed to model racing environments and car behaviours. It is called TORCS (The Open Racing Car Simulator). Each car of the platoon is connected to the environment emulator (TORCS) in order to obtain the relevant “physical” data, like the position and proximity sensor data. Clearly, a car is only allowed to obtain from the emulator information relative to its own sensed physical data (the position and speed of other cars, for instance, can only be obtained by communicating with those cars through the payload communication network). This connection was made through an Ethernet network (emulator network), but could also have been made through some other type of network (even a wireless one).

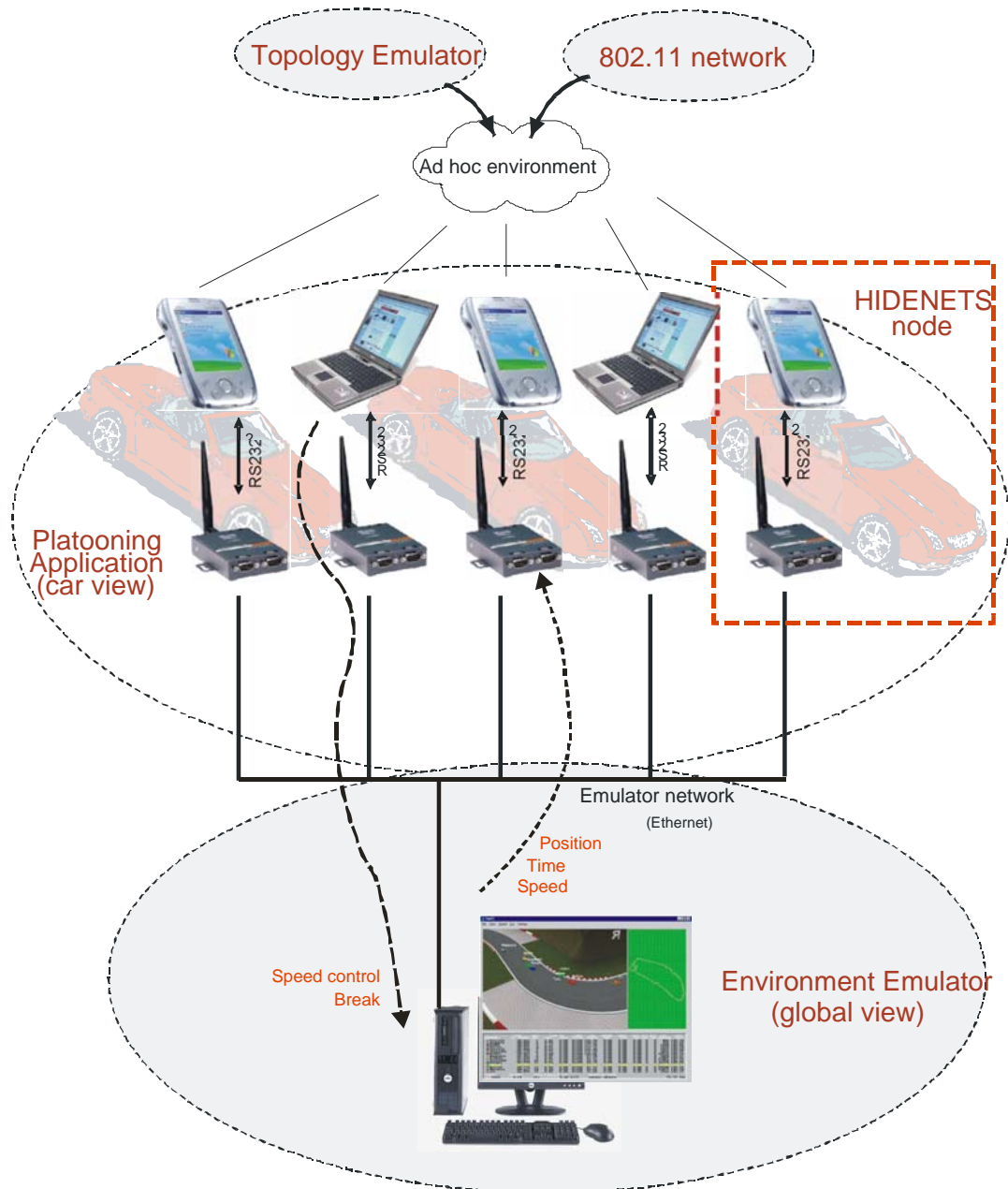
Regarding the ad-hoc communication among the cars in the platoon, it can be implemented using real 802.11 wireless networks. However, this has the limitation that it may be difficult to create controlled perturbations in the ad-hoc network, as necessary to evaluate the behaviour of the application in such situations. Therefore, it may be possible to also emulate this ad-hoc environment, just like we emulate the physical environment. For this, the topology emulator can be used (see section 6.4.3.3). In this case, it will be necessary to connect the “cars”, or more specifically the payload devices, to the topology emulator. Once again, this can be done either through an Ethernet network or through another emulator specific network. In the current implementation of the test-bed the topology emulator is not yet integrated and therefore a 802.11 network is used for the ad-hoc environment. On the other hand, since the network is essentially reliable and always connected, network interferences are simulated by software on the end nodes.

Figure 4-5 provides a concrete perspective of the proof-of-concept set-up.

### 4.3.1 Platooning application architecture

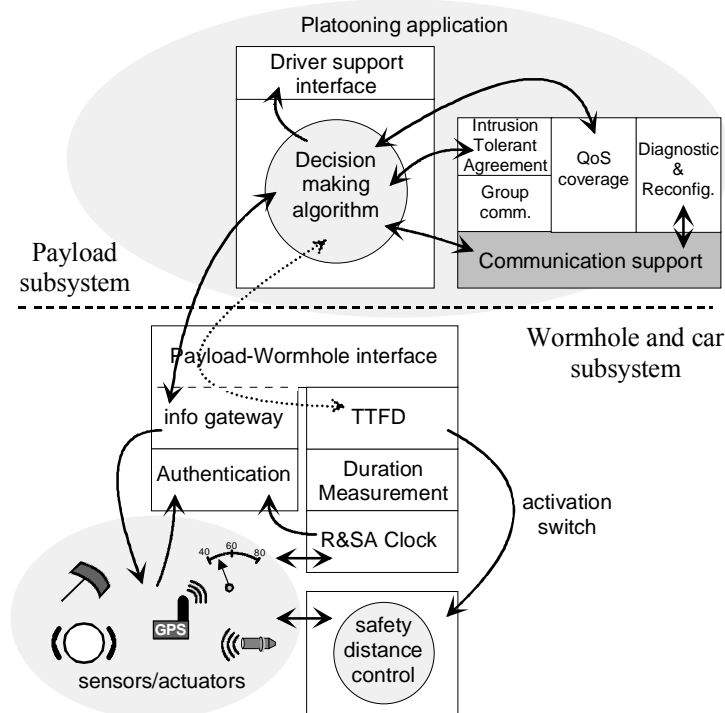
In this section we provide an architectural view of the Platooning application, which will be followed, in the next sections, with descriptions of the main aspects concerning the decision making (control) algorithm, the communication and the interfaces.

Figure 4-6 illustrates the overall architecture of the system for the Platooning application. There are two separate environments, corresponding to the payload subsystem and to the wormhole and car subsystem. The Platooning application central component is the decision making algorithm, executed in the payload part of the system as a typical HIDENETS application. In brief, this algorithm runs in successive rounds, collecting context information from the relevant sources, processing this information, and sending the resulting speed control commands to the car subsystem, through the provided wormhole interfaces. This information, or selected parts of it, may also be provided to the car driver through a convenient interface. In addition, local information that was collected from the car subsystem is sent to the preceding and following cars. Finally, whenever the driver activates the platooning, this will trigger the execution of an agreement protocol to select a new target speed for the platoon, which can be done in parallel with the remaining activities.



**Figure 4-5: Proof-of-concept set-up.**

In the payload part of the system, the Platooning application makes use of communication related services to support its execution. As already mentioned, an **Intrusion Tolerant Agreement** service is used to select a new common target speed for the platoon, which in turn makes use of **Group Communication** functionalities. These services rely on basic communication support that is provided to all applications. Another service that is used is the **QoS coverage** service, which provides awareness about the coverage that can be achieved for some assumed message transmission delay. Since the message transmission delay has a direct impact on the accuracy of the decision making algorithm, it is important to use this information in the calculation of the speed control commands. Finally, a simple **Diagnostic and Reconfiguration Service**, mainly a monitoring service, can be used as an additional service to improve the dependability characteristics of the payload part of the system, in particular the communication services. In the current implementation of the test-bed all these software modules are developed in C and C++, and the system is run on a Windows XP environment.



**Figure 4-6: Platooning application architecture.**

The temporal correctness of the decision making algorithm can be compromised due to several factors, in particular because the execution takes place in the payload subsystem, which is by definition unpredictable with respect to timeliness. Therefore, every step of the execution must be associated with a time limit for its completion, which is monitored with the help of the **Timely Timing Failure Detection (TTFD)** service. If the step is completed within the required deadline, then the Platooning application is considered as timely and speed control commands are accepted and forwarded to the car actuators. As soon as one step is completed, another is immediately started in order to ensure that there is always a time interval being monitored at any time. If the decision making algorithm fails to complete some step within the required deadline, this will be detected by the TTFD service, which will immediately activate the safety distance control component of the car infrastructure. Although possibly more pessimistic (as explained earlier), this will ensure that cars will never collide.

The described behaviour is achieved by means of correctly using the interface provided by the TTFD service, which will be addressed below. The TTFD service makes use of both the **Reliable and Self-Aware Clock (R&SA Clock)** service and the **Duration Measurement** service. This is necessary in order to obtain meaningful timestamps (that are also used by the application, as part of the QoS adaptation service) and also to measure time intervals as necessary to detect timing faults.

The payload-wormhole interface is also used to collect information from car sensors and to send control information to car actuators. An information gateway is required in order to mediate these interactions: the Platooning application cannot directly interact with these devices because they “live” in a real-time subsystem and therefore interactions must be constrained. On the other hand, information collected from the sensors (and time stamped using the R&SA Clock service) can be authenticated using the **Authentication** service. This will ensure that the position information received from other cars is authentic and has not been tampered with.

### 4.3.2 Decision making algorithm

The decision making algorithm can be made arbitrarily complex, just depending on the number of considered variables, which are in direct relation to the degree of detail of the considered environment model. Given the objectives of this proof-of-concept set-up, we tried to simplify as much as possible while keeping the necessary variables to illustrate the use and benefits of the HIDENETS architecture and services.

The considered Platooning application is specified in terms of its functionality and interaction with both other applications (in other cars) and the local wormhole.

The functionality offered by the application is:

- Set target speed – using the information received from other cars and its own instantaneous speed, the application can set the car speed (by sending a new target speed to the wormhole).
- Set validity start – the control algorithm computes a decision based on the assumption that the target speed is set at a given instant. That payload can explicitly set that instant to ensure that the new target speed does not take effect before the planned instant.
- Set deadline – based on available timing and speed information and on the calculated target speed, the application can set the next point in time when new control information must be generated to keep the system safe.

The application requests the following information from the wormhole:

- Position – is provided by the GPS through the wormhole gateway;
- Instantaneous speed – is provided by a car sensor through the wormhole gateway;
- Timestamp – is provided by the R&SA Clock through the wormhole gateway.

The application continually sends/receives the following information to/from the other cars:

- Position;
- Instantaneous speed;
- Timestamp;
- Identifier – was agreed among all cars in the platoon at the start up.

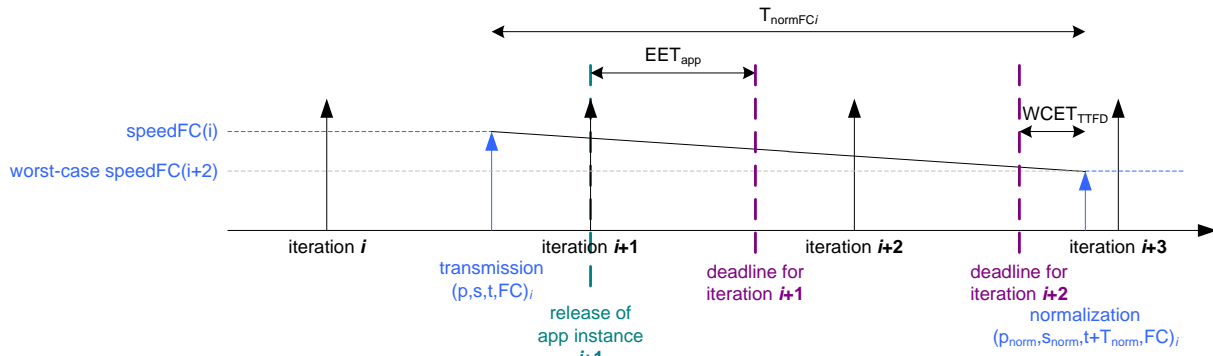
The application sends the following information to the wormhole:

- Target speed,
- Validity start instant,
- Validity end instant (deadline).

The control algorithm is periodic, and functions in either of two modes: boot mode and steady-state mode.

In boot mode there is no initial deadline. The application sets a deadline for the following iteration of the algorithm, but the car remains controlled only by the wormhole, in a safe mode where the safety distance sensor is used. Only if the set deadline is respected does the payload enter in steady-state mode.

To remain in steady-state mode, every period the control task (payload application) wakes up and has to send a command to the wormhole before the previously set deadline expires. This means that the Expected Execution Time of the payload application ( $EET_{app}$ ) must be smaller than the relative deadline set for the iteration. The command sent to the wormhole contains a start instant, a speed value and a deadline for the following iteration. It also defines an effective period for the command, specified as the interval between the start instant and the provided deadline. If the command is received by the wormhole before the start instant, the new speed is accepted and the deadline for the next iteration is set. Then, at the start instant, the wormhole sets the requested speed in a lower level mechanism of the car, which will try to reach that speed until the end of the effective period, by braking or accelerating. This is summarized in Figure 4-7.



**Figure 4-7: Periodic control algorithm.**

The computation of the speed values can be summarized as follows. The payload starts by requesting from the wormhole an update of the GPS and sensor values, hence obtaining its own position and velocity for a given instant. These values and those from the other cars are projected to the normalization instant, assuming the worst cases of the system dynamics. This normalization instant is calculated using the deadline for the following iteration and the Worst Case Execution Time of the Timely Timing Failure Detection ( $WCET_{TTFD}$ ). If the fundamental requirements would be violated at the normalization instant the computed decision will be to brake the car. Otherwise, it computes the speed to set for the new period that better approximates the car to the agreed speed and still respects the fundamental requirements.

### 4.3.3 Communication Aspects

Communication between cars takes place using the available communication services over an emulated ad-hoc wireless network. These are the “standard” communication services available to all possible applications running in the car node, possibly including infotainment or other web based applications. Because of that, communication over this network is subject to unpredictable delays, as discussed in the Fault Analysis section (see section 4.4).

Specific HIDENETS solutions may be incorporated to improve the characteristics of this payload communication, namely in terms of delay, jitter and availability. However, despite improvements these uncertainties are always present and therefore, for the purposes of the Platooning test-bed we will consider a normal payload network. The communication is of a periodic nature. Each car sends information (position, instantaneous speed, timestamp and identifier), called a data set, periodically to other cars.

### 4.3.4 Payload-Wormhole Interfacing Aspects

This section describes a set of relevant details concerning the interaction between the two environments of a Platooning application: the payload subsystem and the wormhole and car subsystems. This interaction is mediated through an **information gateway**. The information gateway implements the specific “logic” of such interaction for the Platooning application. In this sense, the information gateway is specific for each application, such as the Platooning application. In this application context, every interaction between the payload and the wormhole will be handled by the information gateway (see Figure 4-6).

The wormhole subsystem can be seen as a real-time repository of information (similar to a real-time data base) representing the car state (i.e., the car real-time representative). In a real car this state information is in fact collected by the wormhole using complementary resources such as car sensors, real-time networks, body electronics interfaces, etc.

In a HIDENETS system, the car real-time representative may also include relevant application-level, time-consistent context-aware (control) information. Actuation commands sent by the payload to the wormhole can be seen as affecting the state of the real-time representative of the car. That means there is a general safety-related requirement for the continuous monitoring of these commands in the time domain. The interaction of the payload subsystem with the wormhole subsystem must be performed timely.

Since the interaction of the payload with the wormhole is made through the information gateway, it is possible to encapsulate multiple wormhole service invocations within only one interaction. For example,

when the Platooning application sends a new control value to the wormhole, it should also signal the Timely Timing Failure Detection (TTFD) service that the current control step has been completed and it should start a new deadline for the next step. All these actions can be done atomically within the same invocation of an information gateway service, through the payload-wormhole interface.

One advantage of having this information gateway is that admission control mechanisms aimed at enforcing the wormhole temporal properties can be secured at this level. This was done by decoupling the mechanism of accepting new service requests from the execution of the wormhole's services and assuring the first cannot deny the service of the latter. Specifically, the information gateway only processes a limited number of requests from the payload during each iteration. The remaining requests remain in a buffer until the next iteration. If the buffer does not contain a complete request, the information gateway will not wait for its remainder. If the payload sends commands faster than they are processed the buffer will fill up and additional commands are discarded. This is consistent with the assumption that the payload and the payload-wormhole interface can be subject to timing and omission faults. From a hardware perspective, the use of the physical communication medium by the payload is also unable to create a denial of service since the medium is much slower than the wormhole's CPU, and thus unable to create interrupt storms or other problematic conditions.

#### 4.4 Addressing dependability requirements

This section establishes the relevant fault models and discusses the means to address these faults in the context of the Platooning application, that is, with the help of the services included in the application architecture.

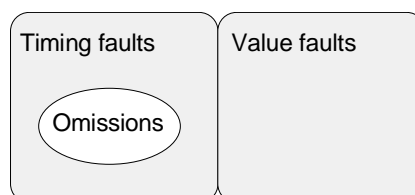
Since the Platooning application architecture (see Figure 4-6) is made up by two separate environments (the payload subsystem running on top of the wormhole and the car subsystem), a specific fault model needs to be considered for each part.

The wormhole and the car subsystem are special parts of the system that are constructed with specific reliability and predictability concerns, thus enjoying a very strong fault model. For the purposes of this test-bed, we will assume that these wormhole subsystems are not subject to faults, otherwise no guarantees can be given about the safety of the car (e.g., if the braking system fails in any way, or if the distance sensor fails, or if the safety distance control module fails, the car is not prevented from colliding).

On the other hand, the payload subsystem is an unpredictable environment that can be subject to a wider range of faults than is the case for the wormhole and car subsystem environment. The software components running in the payload environment (e.g. the HIDENETS middleware services and the Platooning application) are therefore assumed in this test-bed to be subjected to the following faults:

- **Timing faults:** software components have unpredictable execution time and no guarantee exists about their progress, leading to unpredictable delays and omissions;
- **Value faults:** software components may produce incorrect results.

The classification of the above fault classes is such that the two classes are orthogonal, as shown in the following Figure 4-8:



**Figure 4-8: Classes of payload faults.**

Dealing with interfaces between components, end-to-end communication between peers at payload level (e.g. two instances of the Platooning application, each running in a different platoon member) is assumed to be subject to the following faults

- **Timing faults:** messages are unpredictably delayed;
- **Arbitrary faults:** malicious messages can be sent to HIDENETS nodes (e.g., Byzantine faults due to intrusions). However, we assume that only a limited number of nodes,  $f$  out of  $n$ , where  $f < n/3$ , can be malicious.

As for end-to-end communication between peers at wormhole level, it must be noted that this kind of interaction is not needed in the considered Platooning scenario. Therefore it does not make sense to postulate any assumption for this kind of interaction. We can note, however, that in a general case these interactions among wormhole peers should require the communication to preserve the (strong) properties of the local wormhole subsystems, being thus immune to faults affecting the communication. However, in practical wireless communication systems this is a very strong requirement, the coverage of which can only be improved by using very specific solutions, such as modified MAC protocols or temporal redundancy schemes, which we do not cover in this proof-of-concept set-up. Some of these possible mechanisms to improve the dependability characteristics of wireless networks, even if intended for a more generic use, are covered in WP3 and addressed in the Resilient Communication test-bed below.

Regarding the interface between the payload and the wormhole subsystems, what is important to make clear is that any interaction will be subject to the weaker fault model which, in this case, is the fault model assumed for the payload subsystem. Therefore, any wormhole service invocation can be affected by the following faults:

- **Timing faults:** an invocation may be affected by an arbitrary delay;
- **Value faults:** input and output parameters may become accidentally corrupted at the interface.

Since the wormhole and the car subsystem are timely and predictable subsystems assumed to be free from faults, the following components are assumed to be always available, correct and timely:

- the GPS device, giving the correct information about instantaneous speed, position etc., as long as the GPS antenna is able to receive and fix the signals coming from the satellites;
- the proximity sensor, giving the actual distance to the preceding vehicle;
- the speed sensor, giving the actual speed of the car;
- the actuators, which execute exactly the speed control commands they receive from both the Platooning application and the safety distance control subsystem;
- the safety distance control subsystem, which decides when (and how) the breaking subsystem has to be used in order to stop the car in critical situations;
- the HIDENETS services included in the wormhole (e.g. R&SA Clock, Authentication, TTFD, Duration Measurement, etc).

The envisioned payload faults are detected/processed using both the services available in the Wormhole part and other middleware services according to the following criteria:

- payload timing faults (including timing faults resulting from the communication in the payload part) are detected by the **TTFD service**, which is built on top of the **Duration Measurement service** and the **Reliable and Self-Aware Clock Service**;
- to deal with **every** value fault produced at the interface with the payload system, it would be necessary to employ replication strategies for the payload components, which we do not consider here. We assume that the accidental nature of value faults can be handled, in most cases, by the internal car safety system, before they propagate to the car actuators. This is further discussed below;
- communication arbitrary faults are treated by the **Intrusion Tolerant Agreement service**.

The requirements that have to be guaranteed by the overall system are safety, timeliness and security. Despite of the unpredictable execution environment, the Platooning application is hence expected to produce timely and correct driving commands to be forwarded to the car actuators, and the internal safety control system is expected to guarantee the safety of the overall system. The Platooning application is expected to **improve the platoon management** (that is, controlling the distance between platoon members), by using

rich context information received from other cars. At the same time, this improvement must be achieved without compromising any safety requirement (cars must not collide) neither any liveness requirement (cars must still move as fast as possible).

In order to produce driving commands, the Platooning application needs some local and remote information to be available and sufficiently fresh. All received information, local and remote, which is used by the Platooning decision making algorithm, can be made available to the decision making algorithm with arbitrary delays after its production. However, all the information is time stamped by the **R&SA clock service**, and authenticated by the **Authentication service**, and therefore it is possible to create a (trusted) system view reported to accurate time instants. Control decisions (in fact, just the car speed) are achieved based on this view and based on pessimistic assumptions about the behavior of cars after that point in time (for instance, that a car brakes immediately after sending its position and speed). The calculated speed is also consistent with the calculated deadline for the next point in time until which a new control decision must be sent to the wormhole actuators.

It is clear that the Platooning application cannot be designed as a typical real-time application. Part of the application (running on the payload) can be affected by delays in the reception of new data from the other cars, and this must not be critical for the safety of the system. The safety is ensured by designing the application in a way that it uses wormhole services. If delays on the payload part get large enough then the information will not be received on time to allow a control decision to be sent to the car actuators. A loss in the reception of a data packet will have a similar effect as a long message delay. In these cases, the safety of the system is ensured by the wormhole services that detect the timing failures and activate more pessimistic but safe procedures.

In the case of value faults possibly affecting the control values sent to the wormhole, the internal actuation system (we can see all the internal car systems tightly connected with each other) can be augmented with some pre-defined rules in order to verify the validity of the received commands before using them. For instance, when there is a value fault that makes the car increase its speed while the distance to the preceding car is decreasing (the internal safety system may be programmed to keep track of this kind of information), the fault that resulted in a wrong value is detected and is handled by the internal car safety system, which gets in charge of stopping the car if necessary. This kind of rules can be put in place to increase the tolerance to value faults, while still allowing the payload application to control the car using the information received from other vehicles.

In conclusion, we can say that as long as the application is able to produce driving commands and does not suffer timing problems, commands are forwarded to the car actuators, preventing value faults thanks to the presence of the safety distance control mechanism. As explained, most of the timely but wrong commands (in particular all “out of bounds” commands) will not be executed because the internal car safety system has always higher priority than the Platooning application. As long as the TTFD service detects that the car actuators are not receiving timely commands from the application, the car control is passed to the internal car safety distance control subsystem, guaranteeing the safety requirements (the car safety distance control subsystem may pessimistically force the car to stop). Finally, the driver always can take control of the vehicle by direct actuation of the car brakes or the accelerator.

## 4.5 Summary of results

In this section we described the Platooning test-bed, providing a perspective of the main characteristics of the considered Platooning application and its fundamental requirements, describing the implementation and explaining how the requirements were satisfied in face of the assumed fault and system models.

The complete architecture of the Platooning test-bed includes several of the HIDENETS services, with the objective of showing in practice the actual relevance and potential benefits of these services. On the other hand, the architecture encompasses complex middleware services, which are implemented in a generic system, as well as oracle services, implemented within a separate embedded system that constitutes a wormhole.

Most of these services have been implemented, both separately, and in an integrated way in the scope of the Platooning test-bed. Similarly, the wormhole has also been implemented and successfully integrated in the

test-bed. A preliminary demonstration of the Platooning test-bed has been done during the second project review. In this demonstration it was possible to show the following aspects:

- Basic Platooning test-bed: The basic functionality of the platoon of cars was shown, in which they all move with a constant speed and close to each other, and stop when necessary to avoid collisions with the front car. It was also shown that the availability of a wireless ad hoc communication network improves the application by allowing a car in the platoon to have richer information about the platoon and, in particular, to become aware when follower cars slow down or stop (thus allowing preceding cars to also slow down and wait for the slower cars). In a steady state, the decision making algorithm is controlling the car speed using the enriched available information.
- Architectural hybridization using a wormhole: Dedicated embedded hardware was used to implement a wormhole in the test-bed, and was shown to work in the scope of the Platooning application. It was possible to observe this wormhole being used as a “gateway” for the (emulated) car subsystem.
- Timely detection (and reaction) upon payload timing failures: The Timely Timing Failure Detection service has been shown to work within the wormhole, detecting the lack of timeliness of the decision making algorithm and activating a safety distance control algorithm at that time.
- Dependable estimation of communication delays by using input from the QoS Coverage service: Using the QoS Coverage service it was possible to show the impact of degradation in the ad hoc communication environment. An increase in transmission delays is detected by the service and is used by the decision making algorithm to become more conservative and increase the distance between the cars.
- Improved (more optimistic) platooning behavior, while still ensuring the main safety properties: In general, the test-bed allowed the demonstration that it is possible to implement a Platooning application in which the availability of additional information gathered through uncertain or unreliable interfaces (in this case from other cars in the ad hoc network) can be used to improve certain operational characteristics (such as the distance between the cars or the smoothness of car behaviors) while still ensuring safety-critical properties.

Finally, it must be mentioned that some additional implementation work is being done to integrate additional services in the Platooning test-bed, such as the Reliable and Self-Aware Clock and a full implementation of the Intrusion-Tolerant Agreement.

## **5 Distributed Black-Box test-bed**

This chapter presents the Distributed Black-Box (DBB) test-bed (TB), which aims at developing a virtual device, the semantics of which are similar to avionics black-boxes, that tracks cars' history in a way that can be replayed in the event of a car accident. This test-bed is solely based on HIDENETS technology, i.e. it provides an implementation of a secured storage using software building blocks.

Our test-bed relies on standard hardware, namely COTS components, and information is securely stored using replication mechanisms, by means of exchanging positioning data between different cars.

### **5.1 Introduction**

The Distributed Black-Box test-bed is intended to develop a test-bed set-up of the research carried out within WP2 and WP3. Essentially, this test-bed illustrates, using a laboratory set-up, the dependability related protocols and mechanisms developed in the context of the Distributed Black-Box application to ensure data availability and integrity. Actually, the Distributed Black-Box application is a good example of the various services and applications that users can benefit from thanks to car-to-car communication. As a "classical" black-box, its aim is to record informational data, such as: engine / vehicle speed, brake status, throttle position, and even the state of the driver's seat belt switch. As a "smart" black-box, it can also be used for extending the recorded information with contextual information concerning the neighbouring vehicles, possibly the various vehicles involved in an accident. Within this test-bed, we will emulate this setting with programmable vehicles carrying computing devices that implement the Distributed Black-Box. Car accidents will be simulated in order to illustrate the use of a Distributed Black-Box.

### **5.2 Use case and dependability requirements**

#### **Dependability requirements**

Within WP2, algorithms, protocols and suitable mechanisms have been designed and developed during the course of the project, and these building blocks are used to construct a higher level application, namely the collaborative backup of critical data between cars.

Collaborative backup is an important use case in the HIDENETS scenarios and has been carefully chosen for its critical nature to be representative for a number of other HIDENETS use cases. The main use case targeted by the test-bed is the car accident one (see D.1.1 [1]), but most mechanisms involved could apply to other use cases. Conceptually, it involves ad-hoc collaboration of nodes, transfers of critical data, geographical analysis, and trust evaluation.

In the distributed black-box applications, all participating car cooperate to back-up each other's positions, speeds and possibly other critical parameters such as motor speed, brake state, etc. The test-bed operates in two independent phases: in the ad-hoc domain, all cars communicate to ensure replication and dissemination of critical data; in the infrastructure domain, cars send the backed-up data to a secure server, freeing storage space for further back-ups.

With respect to the HIDENETS reference architecture the following relevant components are referenced in this test-bed: proximity map, trust and cooperation oracle, and cooperative data backup.

#### **Application/use case characteristics**

The test-bed set-up described in this chapter concerns the car accident use-case (see D.1.1 [1]), and in particular the Distributed Black-Box application. More generally, the applications that exhibit requirements similar to cooperative data backup are typically mobile (ad-hoc) applications that generate critical and localised data.

#### **Type of faults addressed by the test-bed**

In the distributed black-box test-bed, two types of faults are addressed: crash faults and byzantine fault-tolerance. Byzantine fault tolerance is guaranteed by the trust and cooperation oracle that ensures that only genuine implementation of the test-bed can operate. Indeed, the implementation only ensures the software is genuine by performing random cryptographic challenges on the code to ensure that other parties are running a non modified HIDENETS code.

Crash faults are the main target of the application: when a car crashes, the idea of the distributed black-box application is to use backup data, eventually available on a secure server, to recover critical data emitted before the accident.

### 5.3 Implementation description

#### Hardware platform description

The DBB test-bed is constituted by a moving ad-hoc part and an infrastructure part. A set of mobile devices (laptops) communicates with wireless ad-hoc interfaces and implements the test-bed of the developed algorithms. The infrastructure part contains an access point and a fixed server used both for trusted backup and, for demonstration purposes, capable of displaying and analyzing data.

The experimental test-bed requires a relatively small number of mobile laptops (4) along with the appropriate positioning, communication and securing devices (GPS, wireless communication, smartcard reader, etc.) for the ad-hoc domain, and one PC connected to a wireless access point representing the server in the infrastructure domain.

From the experimental point-of-view, much work was required in order to develop a convincing platform for evaluating a real-world application in a laboratory-sized room. As described below, all components of a typical car-to-car system have been scaled down to reflect this size constraint.

*Infrastructure:* There is an access point through which nodes can connect to the infrastructure network. In the DBB test-bed, the infrastructure network has been simplified and is composed of a trusted server (or a cluster that offers trusted storage) and a displaying device.

*Mobile nodes:* Each node represents a car, and is composed of the following equipments:

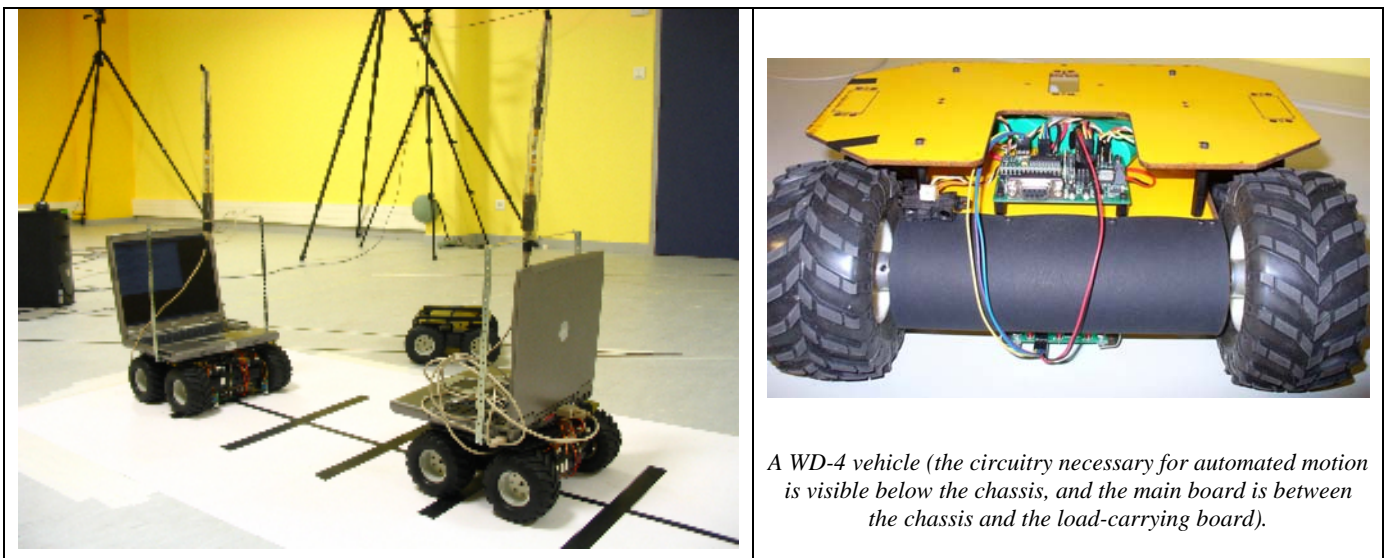
- A **laptop**, which represents the computing elements present in the car. This laptop is running the black-box application, thus using different HIDENETS modules (namely the Trust and Cooperation Oracle, the Cooperative Backup and the Proximity Map modules).
- A **positioning device** that is used by the Proximity Map. In an indoor scenario, a classical GPS-based solution cannot be used. The technology used in this test-bed is the cricket indoor location system [15] the precision of which is sufficient for the needs of the demonstration. This technology uses Ultrasound and RF signals and attains theoretically a centimetre -level accuracy for positioning.
- A **wireless network interface** to connect to the infrastructure. This network interface is a traditional 802.11g card.
- A **smartcard reader**. This device is used by the Trust and Cooperation Oracle to certify that the running software is a genuine one, i.e., is an unmodified version of the HIDENETS platform.
- An **ad-hoc only network interface**. To be able to emulate a car-to-car system, several possibilities to reduce the range of communication have been investigated. The final set-up is composed of a tuneable WiFi interface connected to an antenna via a signal attenuator, the whole system being enclosed in a faraday-cage-type metal grid. Using this hardware system, we showed that it is possible to have ad-hoc communication on short ranges only (typically 1-2 meters, which represents a scale factor of about 20). Figure 5-1 shows a prototype of the modified WiFi interface. The used metal grid is displayed in the bottom part of the picture. A complete set-up is also displayed with, from left to right, a USB connection, a USB WiFi adapter, an attenuator (enclosed in the plastic tape), and an antenna.



**Figure 5-1: HIDENETS reduced-range WiFi interface**

- A **programmable vehicle** capable of carrying the workload (laptop with network interfaces). The experimental set-up uses four-driving-wheels programmable robots that were programmed to follow a black line on the floor. This simple set-up permits to conduct multiple experiments using the same mobility patterns. Figure 5-2 shows such a vehicle carrying a laptop. On the right-hand side, a detailed view of the robot shows the infrared sensors used to follow a black line. On the left-hand side, the complete set-up is shown: two vehicles are carrying the laptop with its network interface on a black line.

In terms of software, we are using Linux with wireless tools. The development of the middleware services is done in Java.

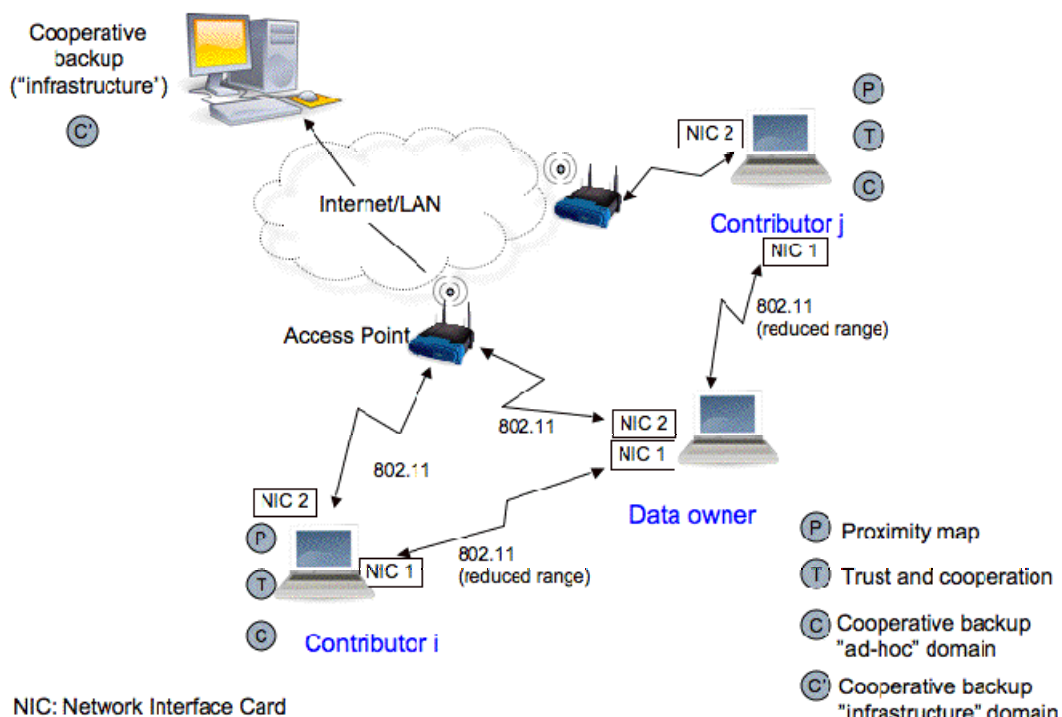


*A WD-4 vehicle (the circuitry necessary for automated motion is visible below the chassis, and the main board is between the chassis and the load-carrying board).*

**Figure 5-2: A complete HIDENETS vehicle (left) and the programmable vehicle (right)**

### **HIDENETS building blocks targeted by the test-bed**

Figure 5-3 shows an overview of the experimental set-up with an identification of the software components and services that are implemented on the different nodes.



**Figure 5-3: Overview of the experimental set-up**

Recall that the testbed operates in two distinct phases:

- In the ad-hoc domain, nodes cooperate to backup data. Data to be backed-up are positioning information taken from the Proximity Map service. To simplify implementation, all communications in the ad-hoc domain are done using a specialized wireless interface, that was reduced in range for feasibility in a laboratory-scale room. In the figure below, this wireless interface is named “NIC 1”.
- In the infrastructure domain, a node sends all the data it has previously backed-up to a secure server, using its second wireless communication device, named “NIC 2”.

On the secure server of the infrastructure, depicted with a desktop computer on the figure, an operator can read backed up information. In the demonstration, the state of traffic can be replayed, with a slight delay that corresponds to the time necessary for nodes to reach an access point of the infrastructure.

The main scenario targeted by the demonstration is the following: three cars are running, backing up each other positions, and sending data periodically to the infrastructure. A car accident occurs, and the positioning information of the cars involved in the accident is destroyed by the crash. After some delay, the third car is able to send the missing information to the infrastructure, permitting the operator to replay the accident and deduct the exact circumstances of the car accident.

Here is a brief description of the various components that we are including in the test-bed set-up:

- **Proximity map:** before being able to backup data, a mobile node has first to discover its neighbours and the corresponding resources. The proximity map represents the local knowledge a node has about its vicinity. This can vary according to wideness (the number of communication hops represented on the map) and according to accuracy (how often is the map updated). Notice that the aim of the proximity map is twofold: it is used to know which nodes can be used for cooperation, and also used as a source of data to be backed up by the distributed black-box application.
- **Trust and cooperation oracle:** in order to interact with unknown neighbours for critical services (e.g., collaborative backup), a node has to evaluate the level of trust it can assign to each of these neighbours. The purpose of the trust and cooperation oracle is to

- i) evaluate this level of trust and
- ii) incite nodes to cooperate with one another.

This module is implemented by verifying that the software running on the node is a genuine HIDENETS module, by using a smartcard-based authentication.

- **Cooperative data backup:** the problem of cooperative backup of critical data, as it can be used for implementing a Distributed Black-Box for instance, consists in four different mechanisms:
  - discovering storage resources in the vicinity,
  - negotiating a contract with the neighbouring cars for the use of their resources,
  - handling a set of data chunks to backup and assigning these chunks to the negotiated resources according to some data encoding scheme and with respect to desired properties like dependability, privacy, confidentiality,
  - and finally it must also take care of the recovery phase, i.e., the data restoration algorithm.

In addition to these middleware components, a simple workload representing an abstraction of the Distributed Black-Box application is used to activate different scenarios that are illustrated during the experiments.

## Scenarios

Various scenarios are being investigated based on the test-bed set-up taking into account for example:

- i) the characteristics of the workload (frequency of data generation, size of the data, temporal validity, *etc.*),
- ii) the data replication strategies, i.e., how the copies are created (full replication of the initial data or fragmentation and replication of the fragments), how many copies are done in the ad-hoc domain and how many need to be available on the fixed infrastructure,
- iii) user mobility scenarios,
- iv) data restoration strategies in the infrastructure domain based on the desired quality of service level, *etc.*

From the communication point of view, it is worth noting that network parameters are radically different if a mobile node is communicating with a neighbouring node or with the infrastructure. For the distributed black-box application, we consider that freshness of the backed-up information is the most important metrics and thus the Cooperative backup service will use only one hop neighbours. Ad-hoc communication is used to disseminate and replicate data, whereas the fixed infrastructure is used as a way for data chunks to reach a permanent storage resource.

## Ad-hoc vs. infrastructure domain

As indicated before, two IEEE 802.11 network interface cards (NIC) are used. The first one is used for the communication in the ad-hoc domain and has been modified in order to reduce its range. Such communication is based on point-to-point interactions<sup>2</sup>. Routing and multi-hop communication is not needed in the context of this test-bed.

Interactions with the fixed infrastructure are done with the second network interface card. Main reasons for this choice are:

- it is assumed that an infrastructure network uses a different network technology,

---

<sup>2</sup> Considering car-to-car communication scenarios, communicating with cars that drive in the same direction will be the main *ad-hoc* interaction scenario: in this case, point-to-point interaction may last long enough to permit both trust negotiation and data transfers (limiting interaction with cars driving in the same direction should disappear with the yet to be defined mobile wireless interface IEEE 802.11P.)

- security concerns are different in car-to-infrastructure or car-to-car interaction (the storage server based in the infrastructure domain is considered a trusted component of the architecture),
- it is assumed that a secure storage system is attainable from the infrastructure.

In the context of the experiment, mobility of the nodes is enforced using two different techniques:

- 1) moving the nodes around the building using programmable vehicles, and
- 2) changing the power transmission parameters of the network interface cards used for ad-hoc communication, which reduces the number of attainable neighbours and thus permits indoor evaluation of car-to-car systems.

Concerning the demonstrated scenarios, and especially fault scenarios, the test-bed demonstrates the use of the distributed black-box application. Several runs will thus show:

- In regular conditions, i.e. with no accident, the moving nodes generate black-box data which is backed-up and pushed towards the infrastructure. The data will be constituted of the nodes' proximity map and some synthetic data representing vehicle speed, position, etc. The infrastructure desktop shows (or replays) the mobility of the nodes, i.e. an aggregation of the nodes' proximity maps. It is worth noting that in real-time mode (during the experiment), the infrastructure desktop will never present an up-to-date representation of the scene, but a scene that is translated according to nodes' mobility and their ability to reach the infrastructure.
- An accident involving two nodes and one or several witnesses. The goal is to demonstrate that the black-box data is backed-up and pushed towards the infrastructure by the witnesses even when an accident occurs. The infrastructure server will be able to replay the accident scenario.

#### **5.4 Summary of preliminary results**

The Distributed Black-Box test-bed is being developed and all design choices, both for hardware and software, have been made and presented above.

The architecture and the building blocks are now available and are being integrated on the hardware platform.

Our tests show that the assumptions on reducing the wireless communication range are valid: a series of experiments using different attenuation values permitted us to choose the right attenuation value for the experiment [16].

The indoor localization service is now tested for its precision. Preliminary tests indicate that the attainable precision is lower than expected.

## **6 Resilient Communication test-bed**

### **6.1 Introduction**

The general idea of this Resilient Communication (RC) test-bed (TB) is a set-up to show the main results of communication-related HIDENETS improvements made in a setting with a mobile ad-hoc network. In particular, the test-bed is placed in the context of car-to-car communication, using the Car Accident use case [1] as an example scenario.

This test-bed is mainly intended as a proof-of-concept of communication-related results from WP2 and WP3. So, the main goal is to allow for experimental research and evaluation on the concepts developed there. In principle, the proposed set-up could also be used for demonstration purposes, although this might require some additional work on presentational issues.

The set-up will be experimental, mainly to evaluate the resilience of HIDENETS solutions in different situations in the ad-hoc domain, such as the occurrence of a wireless link breakdown. Finally, it should be stressed that the experimental set-up will be simple, easy to understand, easy to use and will show the benefits of using resilience mechanisms at communication layers.

The following sections present an overview of the targeted dependability mechanisms, a description of the general set-up and architecture of the test-bed and details on the implementation of the components under evaluation.

### **6.2 Targeted dependability mechanisms**

The WP6 work in this test-bed includes the demonstration of algorithms, protocols and mechanisms at the lower layers of the communication stack. The test-bed will demonstrate the most relevant communication related results from WP2 and WP3 (see D2.1.2 [2] and D3.1.2 [3]):

- Dependability improvements of using a multi-channel / multi-radio architecture in an ad-hoc mesh network.
- Testing of the communication protocols developed with respect to dependability.
- Testing of custom made distributed service replication middleware in the ad-hoc domain.

As a result, the evaluation of the following functionalities is expected from this test-bed:

- Multi-radio MAC layer
- Application of fast reroute techniques in the ad-hoc domain
- Replication manager

The testing and evaluation of the components mentioned above, requires controllable multi-hop topologies. Setting up reproducible experiments with multi-hop topologies based on actual wireless communication is rather difficult. For this reason, a topology emulator tool was developed to allow for reproducible and controllable experiments.

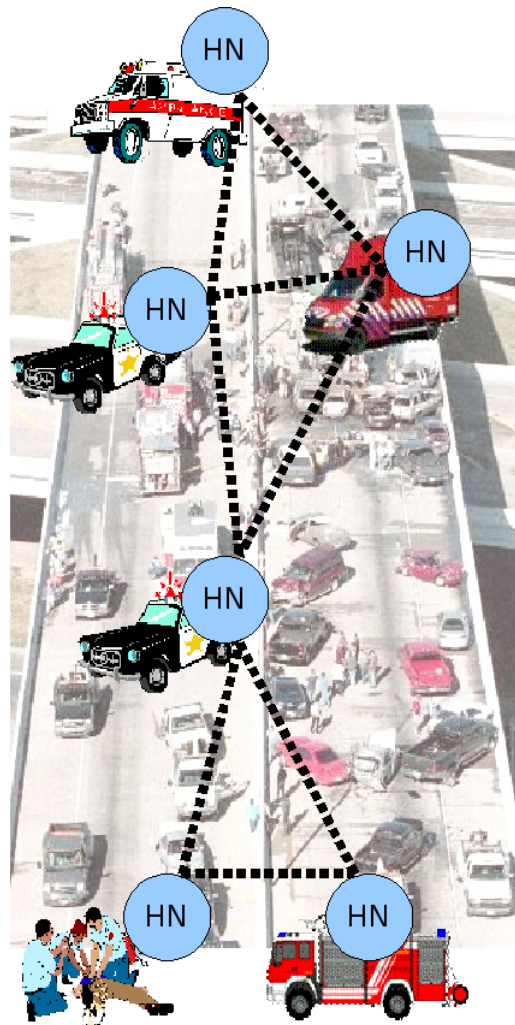
## 6.3 Resilient Communication TB set-up

### 6.3.1 Scenario

As was already mentioned in the introduction of this chapter, the application scenario is based on the Car Accident use case, namely an emergency situation right after an accident in which ambulance personnel on their way to the scene would like to set up a connection with e.g., a policeman/fire-fighter to prepare the treatment of the patient. This connection will be voice/video in first hand, and will also include data transport, with both parties being able to access and change the data. In the case of an accident like a chain collision, local multi-hop communication at the accident scene can be envisioned (see figure 6-1).

Two applications in the context of this example will be used, both providing different functionalities and with different communication level requirements (note that simplified implementations of the applications will be used, as this test-bed focuses on communication-related components):

1. VoIP/Video: a delay-sensitive application, without guaranteed delivery in a multi-hop ad-hoc network (i.e., Audio and video streaming/Data streaming, D1.1, section 3.2.12/3.2.13, [1]).
2. Blackboard (D1.1, section 3.2.9 [1]): an application that is not delay sensitive, however delivery should be guaranteed, with multiple users accessing the data.



**Figure 6-1: Example of a HIDENETS network topology at a chain collision accident scene**

### 6.3.2 Network architecture

The RC test-bed described in this chapter is set up with a limited number of ad-hoc nodes that are based on the algorithms and protocols developed within WP2 and WP3. These devices constitute an ad-hoc network to show the effect of resilience mechanisms in the ad-hoc domain. An example of a network topology was shown in Figure 6-1 in the previous section.

### 6.3.3 Node Architecture

The architecture of nodes used in the test-bed implementation (see Figure 6-2) is based on the HIDENETS node-architecture, but focuses on the following communication-related components:

- Multi-Channel MAC layer
 

This is a MAC extension layer that is implemented on top of the network interfaces (hardware) to support nodes with multiple (wireless, IEEE 802.11a) interface cards are used. Towards the network layer, the multi-radio MAC layer looks like a single MAC device.
- Channel selection
 

When using multiple channels, it is important with respect to the dependability of the system to have a robust channel selection algorithm. This component assigns frequency channels to the multiple physical interfaces on a node.
- Fast reroute techniques
 

In WP3, we developed and evaluated a set of fast re-route techniques for the IP communication. One of these is implemented in the test-bed.
- Replication manager
 

For application level replication, the replication manager is used. Replication needs timely and correct delivery of packets. The ability of the HIDENETS communication services to fulfil this requirement demonstrates the advantages of the HIDENETS middleware.

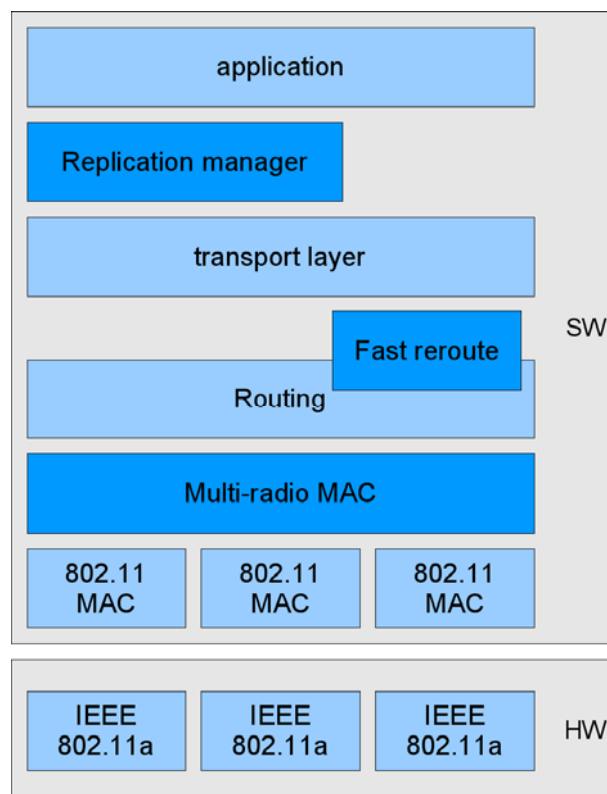


Figure 6-2: Resilient Communication software architecture

### 6.3.4 Topology emulation

Depending on the component under evaluation, the set-up may be slightly different. For instance, while in some cases it is required to have real wireless communication. In other cases, using real wireless communication is not the optimal way to evaluate the concepts. In a wireless setting, it is very hard to control network topology and perform reproducible tests. A solution for this problem is to emulate the wireless network, see 6.4.3.3.

## 6.4 Implementation details

### 6.4.1 Multi-Channel Multi-Radio

Using a multi-channel multi-radio architecture improves the performance in ad-hoc networks, especially in dense multi-hop networks. Using multiple channels reduces the interference in multi-hop communication, improving both throughput and end-to-end delay; using multiple radios per node increases the performance gain and allows for using multiple channels without losing connectivity between nodes. The assignment of channels to radios in a distributed way is crucial in optimizing the performance while guaranteeing connectivity.

In the remainder of this section, details on the implementation of the HIDENETS multi-channel multi-radio architecture proposed in WP3 [3] will be presented. This architecture allows for a robust, broadband ad-hoc mesh network.

#### 6.4.1.1 Hardware implementation

The use of multiple radios makes it hard to use laptops or PDA's, as these mostly do not support 3 radio interfaces. So for the multi-radio scenario, the ad-hoc node will look slightly different; the multiple radios will be implemented in a separate device (i.e., an embedded platform). The multi-radio hardware that is used is shown in Figure 6-3. The hardware is based on an i386 architecture, the multi-radio part consists of three IEEE 802.11a interfaces (miniPCI Ubiquity SR5 cards).



**Figure 6-3: A multi-radio node**

The multi-radio hardware solution could be used as in-car communication devices. In this case, applications could also run on the embedded device, but for testing and demonstration purposes this is currently not the case. A laptop will be connected to this device via an Ethernet connection to control the device and run applications. Besides providing a multi-radio mesh interface, the hardware is extendible with other

technologies currently not included in this test-bed. For instance, it could serve as a gateway to infrastructure networks by providing access to IEEE 802.11b/g hotspots or UMTS networks.

So the multi-radio devices will form a multi-radio multi-channel ad-hoc mesh network. The laptops are used to run software for performance evaluation, measurements and demo applications.

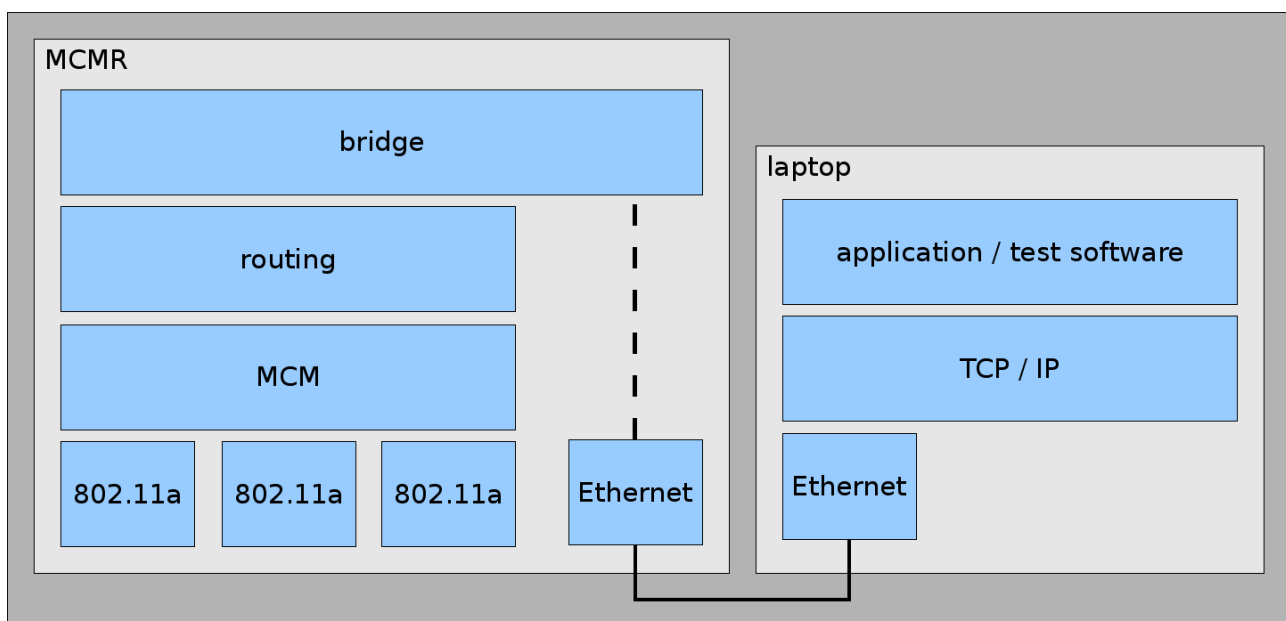
#### 6.4.1.2 Software components

The multi-radio hardware runs Voyage Linux and uses the MadWiFi IEEE 802.11 open source drivers in ad-hoc mode. To allow the Ethernet connection to a laptop, the Ethernet port is bridged with the Multi-Channel MAC layer (MCM); i.e., layer 3 and above are implemented in the laptop. For the multi-radio tests, a layer 2.5 routing module is used to eliminate any possible negative influence of bridging to the laptop.

The multi-radio implementation consists of multiple COTS IEEE 802.11a interfaces and the MCM layer that combines these into a single layer 2 interface. More details on the MCM layer can be found in [3]. Although there is an implementation of the (3,6)-algorithm proposed in WP3, in the test-bed a fixed channel assignment is used to allow for controllable and reproducible experiments.

The Linux distribution Ubuntu [17] is used on the laptop. The laptop is used to control the multi-radio hardware using an SSH connection, and to run measurement software and the video application. For performance evaluation *iperf* and *ping* are used to evaluate throughput and delay.

A component view of a multi-radio node (consisting of a multi-radio device and a laptop) is shown in Figure 6-4.



**Figure 6-4: Multi-radio node architecture**

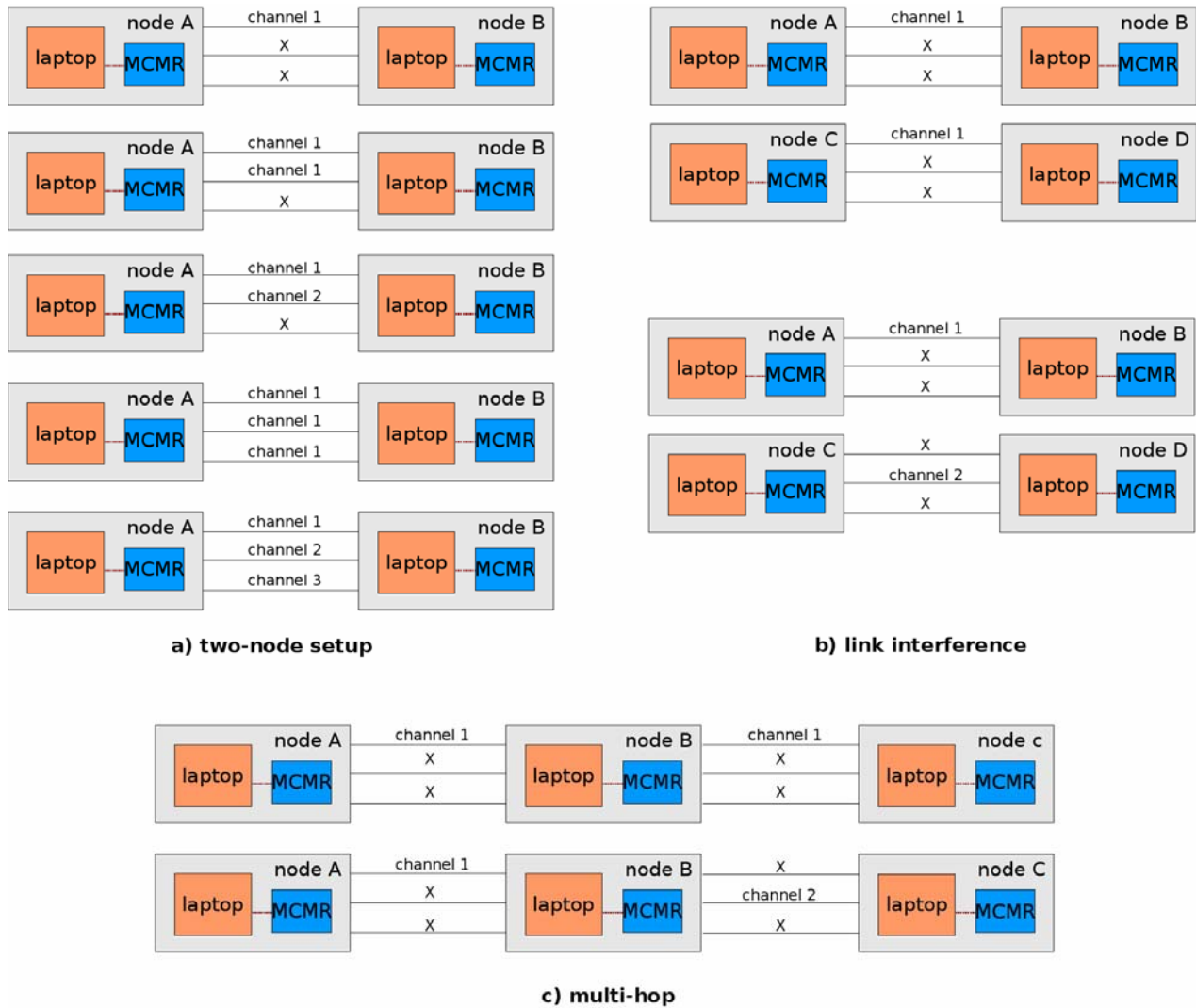
#### 6.4.1.3 Test/evaluation scenarios

The multi-radio implementation will be evaluated using the implementation of a multi-radio node described before in a setting with a limited number of nodes. The goal of this evaluation is to evaluate the performance gain that is expected based on theoretical work in WP3. Basically, multi-radio is expected to improve performance based on:

- reducing interference between nearby links (i.e., between link AB and link CD, see Figure 6-5)
- reducing interference in multi-hop communication (i.e., between link AB and link BC, see Figure 6-5)

This will be evaluated by measuring performance characteristics (i.e., throughput and latency) in both single-radio and multi-radio cases using the network in Figure 6-5. Also, an evaluation will be made of the internal interference between physical radio interfaces on one node as initial tests have shown this has a significant impact on the performance. Test scenarios that will be used for performance evaluation are shown in Figure 6-5.

Besides evaluation of the multi-radio implementation and the performance testing that will be done, a simple demonstration will be implemented that will show the main benefits of using multi-radio mesh interfaces using a video application. The exact implementation will depend on the results of the initial evaluation and testing.



**Figure 6-5 Multi-channel multi-radio test scenarios**

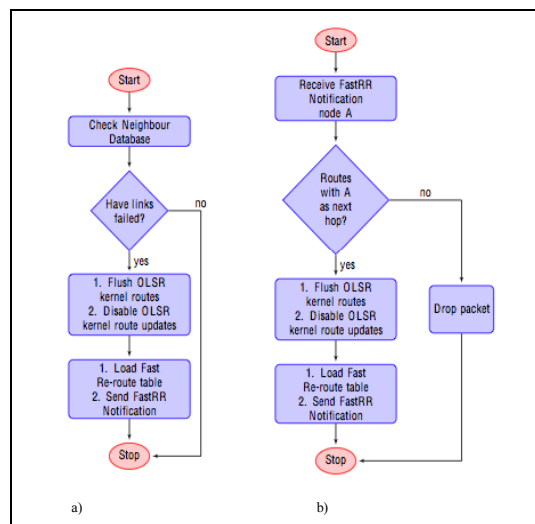
## 6.4.2 Fast Reroute

Wireless networks are very prone to failures, and it is very important to support fast rerouting of critical HIDENETS applications. In WP3 [3], we have evaluated a large set of fast reroute schemes. This evaluation pointed at DMRC [13] as one of the best candidates to pre-plan backup next hops during failures. A complete fast reroute scheme should also address link failure detection and failure notification. In the following we describe the implementation of this complete scheme on a UNIX © platform.

The Fast Reroute (FR) scheme that is implemented consists of two elements, a Link Failure Detection (LFD) mechanism and a Fast Reroute Notification protocol (FRN). The performance of the Fast Reroute scheme is tightly dependent on the effectiveness of the LFD mechanism. Detection of link failures in radio broadcast networks can be implemented in various ways, e.g., using the received signal strength from one-hop neighbours as an indicator of the probability of link failures, or/and using some kind of beacon signal where a loss of one or more beacons is an indication of a link failing.

### 6.4.2.1 Implementation

The FR mechanism is implemented as an extension to the Optimized Link State Routing (OLSR) protocol [12], executing as a daemon in user space and is compatible with various UNIX © platforms. The FR requires exclusively access to the kernel routes. The outline for the implementation is illustrated in Figure 6-6.



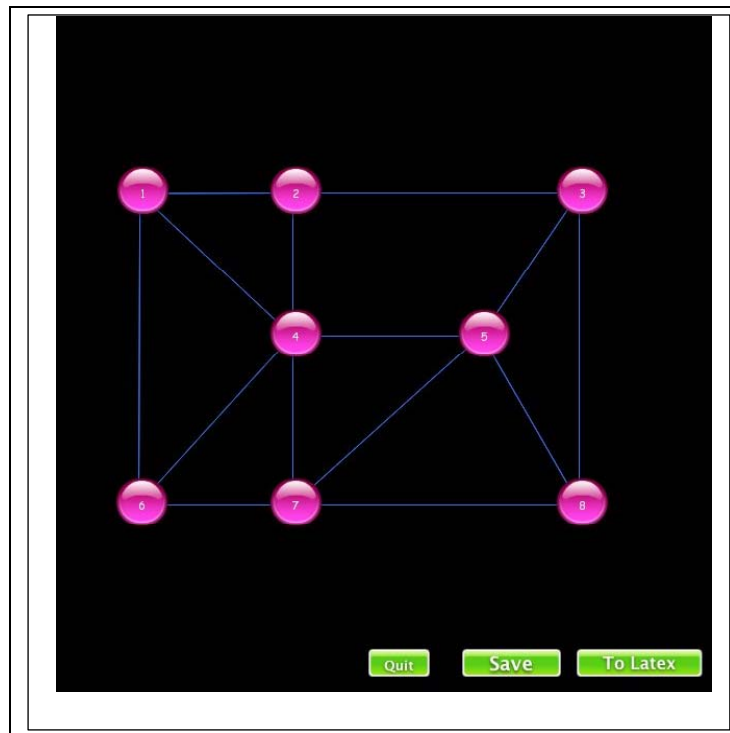
**Figure 6-6: Fast Reroute Notification implementation**

### 6.4.2.2 Evaluation

As the number of nodes in a network increases, it gets increasingly unpractical to evaluate the implementation using any kind of laptop/hardware platform. In order to test and evaluate the implementation we use a software switch which can execute on any UNIX ® platform. The switch has a command line user interface where one can define a set of nodes and the connectivity between them. The switch is limited to basic functionality such as adding/removing nodes/links and displaying routing tables for the different nodes. The only traffic flowing between the nodes is the routing traffic generated by OLSR and the FR daemon.

Since the user interface to the switch is not very visual, we have implemented a graphical front-end (GUI) to the switch where the nodes and the connectivity defined in the switch interface are displayed. As links are removed or added they are updated in the graphical interface. When FR is activated, the color of the links connecting the nodes affected by this is changed, creating a visual effect when a link fails.

The GUI is implemented in the Python scripting language, using its game API to create the graphical display. The software switch communicated with the GUI through a shared memory segment.



**Figure 6-7: The GUI for the software switch**

### 6.4.3 Replication Manager

In order to increase availability and reliability of state-full applications, redundancy as provided by replication in cluster solutions is a well-known and frequently utilized approach. For mobile services in dynamic ad-hoc networks, such replication mechanisms have to be adapted to deal with the frequently higher communication delays and with the intermittent connectivity. Dynamic clustering strategies in which the replica set is adjusted to the current network state can help to handle the network dynamicity.

Larger communication delays can increase the probability of inconsistent replica state, so that dynamic cluster member selection can lead to substantial improvements in mobile scenarios. The replication manager (RM) defined in D2.1.2, is an example of such a middleware component that provides applications with a resilient shared memory area and performs management of such a dynamic cluster.

#### 6.4.3.1 Software components

The replication manager consists of a Replication Manager daemon that runs in the background on each of the servers in the cluster. The Replication Manager announces itself via the Zeroconf [18] protocol to neighbouring nodes in the ad-hoc network. Towards the server application the replication manager provides a Dbus interface via Glib bindings [19]. The replication manager is developed on Linux and written entirely in C. It is using the Glib object oriented library and Dbus for IPC and for RPCs as well. Furthermore it uses the Avahi implementation of the Zeroconf protocol [20].

#### 6.4.3.2 Test/evaluation scenarios

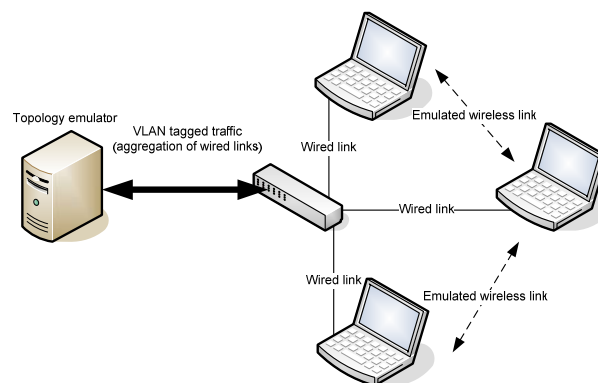
The replication manager will be tested using the topology emulator. We will be using an emulated IEEE 802.11a technology. We will perform tests with different standard random mobility models e.g., Brownian motion and random waypoint mobility. Furthermore we will set up a scenario to test the replica selection mechanisms where two nodes travel back and forth closer to and farther away from the cluster and then these nodes should be exchanged in the group accordingly.

#### 6.4.3.3 Topology emulator

The objective of the topology emulator is to provide an emulation test-bed for wireless applications, where the wireless link is replaced by a wired link. The wired link is less exposed to uncontrollable disturbances

and thus more controllable than a wireless link. The properties of a wireless link, such as packet drops and delay, are then enforced on network traffic on the wired link in a controllable and reproducible manner based on simulations.

The emulator node receives all frames transmitted between end-nodes. No end-nodes receive frames before they have been bridged by the emulator node. Node separation is obtained through IEEE 802.11q virtual LAN (VLAN) tagging on the switch. This concentrates all frames from one end-node into one VLAN unique for that end-node. The VLAN-tag is also used by the emulator node to identify the sources of the frames. By enabling the emulator node to bridge between all virtual LANs, all end-nodes can successfully transmit frames to each other, given they are forwarded by the emulator node. By selectively dropping or delaying frames, the emulator node controls the link properties of all links between end-nodes. Ultimately, as the emulator node forwards frames, its existence is by design transparent to any networking protocols (Layer 3 and higher) used on the end-nodes. The architecture of the topology emulator is illustrated in Figure 6-8.



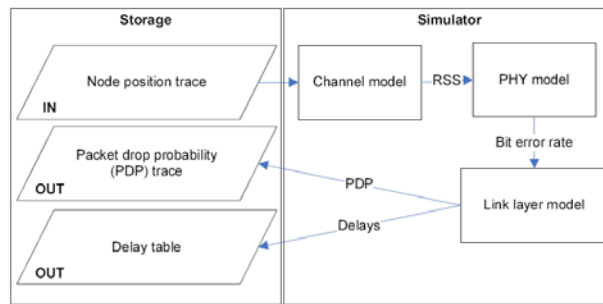
**Figure 6-8: General architecture of topology emulator. The emulator node supports connecting up to 20 end-nodes to the switch.**

#### 6.4.3.3.1 Use case in Resilient Communication test-bed

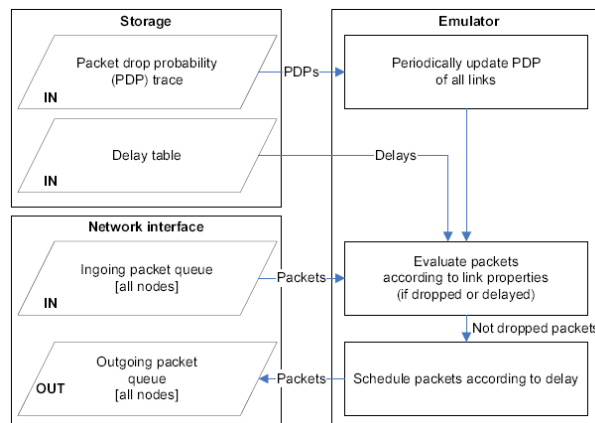
For this particular test-bed, the topology emulator is introduced as to provide for testing in larger networks. Using real wireless components when testing the replication manager in a large network will require many available wireless platforms, immense coordination of the platforms to create the desired scenario and additionally, introduce unnecessary sources of errors in terms of uncontrollable packet losses and link instabilities. Instead, the topology emulator is used to create a wireless network when testing the replication manager by emulating the properties of the network centrally and applying this to the ongoing traffic during the evaluation.

To use the topology emulator for evaluation in RC TB, several types of input are needed for simulation and emulation. The simulation process is depicted in Figure 6-9 and the emulation process in Figure 6-10. Two kinds of simulation models are used; 1) a mobility model and 2) a communication technology model as well as traffic input from the application under test. The mobility model is reflected in the topology emulator as a movement trace, i.e., a file containing recorded positions of all nodes over the entire time-period of the test execution. The communication technology model is used to transfer the node movement into communication properties, such as packet drop probability (PDP) and delay which are then emulated onto the application traffic. For the replication manager the Brownian motion or random waypoint mobility models are used. For the communication model, a detailed model of IEEE 802.11a is used, including shadowing and multi-path fading properties. The details of the communication model are described in [1].

The replication manager is installed and run on every end-node. During evaluation, the topology emulator is transparent to the end-node applications. The only presence that can be utilized is the recording feature of the topology emulator which records every packet forwarded through the emulator. This feature can be used when evaluating results to track packet drops and link changes. Any additional coordination, such as initiation of test execution and synchronized recording of results within the test-framework is set up on the end-nodes using scripts.



**Figure 6-9: Illustration of the simulation process of the topology emulator.**



**Figure 6-10: Illustration of the emulation process of the topology emulator.**

#### 6.4.3.3.2 Deployment

All end-nodes in the topology emulator are connected to a Cisco Catalyst 2950T VLAN-aware switch, equipped with two Gigabit ports. One of these ports is used in trunking mode, meaning that all traffic sent on other ports is forwarded to this port. The emulator node is then connected to one Gigabit port for forwarding packets using Iptables. The machine has an Intel Core 2 Duo 1.86GHz processor and has Linux kernel v2.6.18 installed.

## 6.5 Summary of results

The Resilient Communication test-bed shows the main results of communication-related HIDENETS results in the context of car-to-car communication. In this chapter, implementation details of the test-bed have been presented, showing the final set-up that will be used for evaluation and testing of the selected functionalities from WP2 and WP3. The following functionalities are implemented in the test-bed:

- **Multi-Channel Multi-Radio**  
This was achieved by using dedicated embedded hardware, which allows for the use of multiple COTS IEEE 802.11a interfaces. An implementation of the Multi-Channel MAC layer is used to manage multiple interfaces and providing a standard communication interface to upper layers.
- **Fast Reroute**  
The Fast Reroute (FR) scheme that is implemented consists of two elements, a Link Failure Detection (LFD) mechanism and a Fast Reroute Notification protocol (FRN). The FR mechanism is implemented as an extension to OLSR, and is compatible with various UNIX © platforms. For evaluation purposes, a graphical front-end was developed.
- **Replication Manager**  
The replication manager consists of a Replication Manager daemon that runs in the background on each of the servers in the cluster. Standard protocols and interfaces are used in the implementation. The topology emulator that was developed will be used for evaluation of the Replication Manager. The objective of the topology emulator is to provide an emulation test-bed for wireless applications

In the remainder of the HIDENETS project, these three components will be further evaluated and tested. Results of this and further documentation on the test-bed will be reported in D6.4 at the end of the project.

## **7 Summary and outlook**

Considering the HIDENETS project as a whole, the resulting solutions are expected to contribute to a user perception of trustworthiness of future wireless services, as this perception is strongly impacted by availability and resilience aspects. Such perception is critical for the technical and business success of these services. The solution development and analysis require a holistic approach combining aspects of communication, middleware, service deployment and access. The WP6 contribution to the final results of HIDENETS embodies a proof-of-concept of the design, implementation and evaluation of resilient solutions for new mobility-aware distributed applications, with critical requirements on open communication infrastructures. The solutions that are developed in HIDENETS follow a holistic end-to-end approach. This deliverable accompanies the test-bed implementations and hence contains a description of the set-up and implementation choices.

In order to focus on certain essential functionalities and to show the benefits of a flexible and modular HIDENETS architecture, four focus areas have been identified which lead to the definition of so-called specialised test-beds. This partitioning allows for the development of stand-alone test-beds that reflect relevant results of related project tasks that can be tested independently in an isolated environment at local sites. As such, these individual four test-beds already represent a set of components that have been integrated using HIDENETS interfaces.

The Application Development test-bed focuses on modelling and design methodologies in the context of a highly available application using a COTS platform based on SAF standards. The Platooning Application test-bed deals with the architectural hybridisation and the implementation of resilient middleware in the context of the Platooning application. The environment that is envisaged is composed of both infrastructure and ad-hoc based communication networks, and will also be wormhole-enabled. The Distributed Black-Box test-bed will essentially illustrate the algorithms, protocols and suitable mechanisms that are designed and developed for collaborative backup of critical data among cars. Finally, the Resilient Communication test-bed deals with the demonstration of resilience communication-related protocols in the ad-hoc domain. This mainly involves their impact on higher protocol layers, in order to cover failures such as wireless link breakdown, node breakdown and congestion. The set-up will be using laptops or multi-radio devices, employing optimised communication protocols. Apart from the four test-beds, also a set-up of a topology emulator is considered. This is used in the Resilient Communication test-bed.

During the course of the HIDENETS project, the different stages of the test-beds have been described in WP6 milestone documents and deliverables. The final versions of the HIDENETS test-beds have been collected in the current deliverable. This implementation will be evaluated and further documented in D6.4 (due Dec 31<sup>st</sup>, 2008).

## **8 List of Acronyms**

AIS	Application Interface Specification
AMF	Availability Management Framework
AP	Access Point
ARP	Address Resolution Protocol
BT	BlueTooth
C2C	Car to car
CGL	Carrier Grade Linux
CL	Cross-Layer
COTS	Commercial off-the-shelf
CPU	Central Processing Unit
DBMS	Database Management System
DM	Diagnostic Manager
DMRC	Multiple Routing Configurations with Deflection
DSE	Domain Specific Editor
FCD	Floating Car Data application
FR	Fast Reroute
FRN	Fast Reroute Notification
GMF	Graphical Modelling Framework
GPS	Global Positioning System
GRM	General Resource Modelling
GSM	Global System for Mobile communications
HA	High Availability
HzW	Hazard Warnings and Information from other vehicles application
HW	Hardware
IP	Internet Protocol
J2ME	Java 2 Micro Edition
J2SE	Java 2.0 Standard Edition
JVM	Java Virtual Machine
LAN	Local Area Network
LFD	Link Failure Detection
MAC	Medium Access Control
MCM	Multi-Channel MAC
MCMR	Multi-Channel Multi-Radio
MSU	Maintenance / Software Updates (application)
NIC	Network Interface Card
OLSR	Optimised Link State Routing

---

OMG	Object Management Group
PDA	Personal Digital Assistant
PDSS	Platoon Driver Support Software
QoS	Quality of Service
RHEL	Red Head Enterprise Linux
RM	Replication Manager
RM	Reconfiguration Manager
RTP	Real-Time Transport Protocol
RTP	Resilient Telco Platform (Fujitsu Siemens product)
R&SA	Reliable & Self-Aware (Clock)
SAF	Service Availability Forum
SLES	Suse Linux Enterprise Server
SW	Software
TTFD	Timely Timing Failure Detection
UML	Unified Modelling Language
UTC	Universal Time Coordinated
UWB	Ultra-Wideband
VANET	Vehicular Ad-hoc Network
VLAN	Virtual LAN
WLAN	Wireless Local Area Network

## 9 References

- [1] M. Radimirsch et al, Use case scenarios and preliminary reference model, EU FP6 IST project HIDENETS, deliverable D1.1, September 2006.
- [2] A. Casimiro et al, Resilient architecture, EU PF6 IST project HIDENETS, deliverable D2.1.2, December 2007.
- [3] I.-E. Svinnset et al, Report on resilient topologies and routing, EU PF6 FP6 IST project HIDENETS, deliverable D3.1.2, June 2008.
- [4] P. Lollini et al, Evaluation methodologies, techniques and tools, EU PF6 FP6 IST project HIDENETS, deliverable D4.1.2, December 2007.
- [5] András Kövi et al, UML profile and design patterns library (preliminary version), EU PF6 FP6 IST project HIDENETS, deliverable D5.1, March 2007.
- [6] I.C.C. de Bruin et al, Specification HIDENETS laboratory set-up scenario and components, EU PF6 FP6 IST project HIDENETS, deliverable 6.2, October 2007.
- [7] RTP: Resilient Telco Platform, Fujitsu Siemens Computers, URL: <http://www.SAFE4CS.com> (information on RTP: Resilient Telco Platform, a Fujitsu Siemens product)
- [8] Service Availability Forum, URL: <http://www.saforum.org/> (Service Availability Forum)
- [9] A. Nickelsen, J. Grønbaek, "Probabilistic fault detection in network communication", Tech. report, AAU, 2006
- [10] Intel Architecture, 32-bit, URL: <http://developer.intel.com/products/processor/manuals/index.htm>
- [11] A. Nickelsen, M. Jensen, E. Matthiesen, HP Schwefel, "Scalable emulation of dynamic multi-hop topologies", 4th International Conference on Wireless and Mobile Communications (ICWMC), August 2008.
- [12] T. Clausen et al, "Optimized Link State Routing Protocol (OLSR)", RFC 3626, October 2003
- [13] A. Kvalbein, A.F. Hansen, T. Čičić, S. Gjessing, and O. Lysne, "Fast IP network recovery using multiple routing configurations", IEEE INFOCOM, 2007
- [14] OpenAIS Standard Based Cluster Framework, URL: <http://www.openais.org/>
- [15] Cricket indoor location system, URL: <http://cricket.csail.mit.edu/>
- [16] M.-O. Killijian, D. Powell, M. Roy and G. Séver, "Experimental Evaluation of Ubiquitous Systems. Why and how to reduce WiFi communication range", - M.-O. Killijian, D. Powell, M. Roy and G. Séverac. Distributed Event-Based Systems (DEBS 2008), Roma, July 2008
- [17] Ubuntu Operating System, URL: <http://www.ubuntu.com/>
- [18] IETF Zeroconf Working Group, URL: <http://www.zeroconf.org/>
- [19] D-Bus, URL: <http://www.freedesktop.org/wiki/Software/dbus/>
- [20] Avahi, URL: <http://avahi.org/>
- [21] Lorenzo Falai et al, "Mechanisms to provide strict dependability and real-time requirements", EU PF6 FP6 IST project HIDENETS, deliverable D3.3, June 2008.
- [22]. António Casimiro et al, "Service level resilience solutions for the infrastructure domain", EU PF6 FP6 IST project HIDENETS, deliverable D2.2, June 2008.
- [23] António Casimiro et al, "Service level resilience solutions for the ad-hoc domain", EU PF6 FP6 IST project HIDENETS, deliverable D2.3, June 2008.