

**Project no.:** IST-FP6-STREP- 26979  
**Project full title:** Highly dependable ip-based networks and services  
**Project Acronym:** HIDENETS  
**Deliverable no.:** D2.2  
**Title of the deliverable:** Service level resilience solutions for the infrastructure domain

<b>Contractual Date of Delivery to the CEC:</b>	30 <sup>th</sup> June 2008	
<b>Actual Date of Delivery to the CEC:</b>	27 <sup>th</sup> June 2008	
<b>Organisation name of lead contractor for this deliverable</b>	FCUL	
<b>Author(s)/ Participant(s):</b>	António Casimiro (editor), Jesper Grønbæk, András Kovi, Anders Nickelsen, Hans Reiser, Thibault Renier, Hans-Peter Schwefel	
<b>Work package contributing to the deliverable:</b>	WP2	
<b>Nature:</b>	R	
<b>Version:</b>	4.0	
<b>Total number of pages:</b>	38	
<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:**

The main objectives of WP2 are to define a resilient architecture and to develop a range of middleware solutions (i.e. algorithms, protocols, services) to address resilience requirements in the design of highly available, reliable and trustworthy distributed solutions. This deliverable presents research results concerning the development of middleware services within the HIDENETS environment, the objective of which is to facilitate the construction of resilient, dependable car2car applications operating in ad-hoc environments in cooperation with infrastructure based services.

The deliverable complements the work presented in deliverable D2.3 (Service level resilience solutions for the ad-hoc domain), focusing on ideas that typically reflect improvements of the services presented in D2.3. These improvements can be achieved due to the possibility of operating over infrastructure environments, in addition to the exclusive operation over ad-hoc environments.

The deliverable provides three contributions. Firstly, it addresses the problem of fault detection from the perspective of end-to-end services, which is addressed under a probabilistic scope and which is suitable for IP based communication systems in general. Then, it provides a study on the dependability/performance trade-off for replicated servers in the infrastructure domain. Finally, an extension to the Intrusion Tolerant Agreement service is introduced, which exploits the assumed availability of a reliable server in the infrastructure domain to improve the performance of the basic (non-extended) service, as presented in D2.3.

**Version information**

Version	Date	Comments
1.0	23.05.2008	Full draft, including all the received contributions.
2.0	02.06.2008	Initial draft, with revision marks from the internal review team
3.0	12.06.2008	Added new input from BME. Made changes according to comments from the internal review team.
4.0	23.06.2008	Made changes according to comments from the external review team.

# Table of Contents

<b>1</b>	<b>EXECUTIVE SUMMARY .....</b>	<b>5</b>
<b>2</b>	<b>MIDDLEWARE RESILIENCE SOLUTIONS.....</b>	<b>6</b>
2.1	OPERATION DOMAINS IN HIDENETS .....	6
2.2	ARCHITECTURAL APPROACH.....	7
2.3	HIDENETS NODE SOFTWARE ARCHITECTURE .....	9
2.4	COMPLEX RESILIENCE SERVICES.....	11
<b>3</b>	<b>PROBABILISTIC FAULT DETECTION USING CROSS-LAYER OBSERVATIONS .....</b>	<b>12</b>
3.1	INTRODUCTION.....	12
3.2	FAULT DETECTION AND –DIAGNOSIS FRAMEWORK .....	12
3.3	BACKGROUND ON BAYESIAN NETWORKS .....	13
3.4	BAYESIAN NETWORK STRUCTURE FOR THE FRAMEWORK .....	14
3.5	PARAMETERIZATION OF THE FDD COMPONENT .....	18
3.6	THE BAYESIAN NETWORK IN THE HIDENETS DIAGNOSTICS MANAGER.....	20
<b>4</b>	<b>OPTIMISED DEPENDABILITY/PERFORMANCE TRADE-OFF FOR REPLICATED SERVERS IN THE INFRASTRUCTURE DOMAIN.....</b>	<b>22</b>
4.1	INTRODUCTION.....	22
4.2	SUMMARY OF THE SIMULATION WORK .....	22
4.3	SIMULATION MODEL APPLICATION .....	24
4.4	HIDENETS-SPECIFIC IMPLEMENTATION.....	26
4.5	SUMMARY .....	28
<b>5</b>	<b>INFRASTRUCTURE-ASSISTED INTRUSION TOLERANT AGREEMENT .....</b>	<b>30</b>
5.1	INTRODUCTION.....	30
5.2	RELATED WORK.....	31
5.3	SYSTEM MODEL .....	31
5.4	DESIGN OPTIONS .....	32
5.5	SUMMARY .....	35
<b>6</b>	<b>CONCLUDING REMARKS.....</b>	<b>36</b>
	<b>REFERENCES .....</b>	<b>37</b>

# **1 Executive Summary**

## **Objectives of the deliverable**

As stated in the HIDENETS Technical Annex, the main objectives of WP2 are “*to define a resilient architecture and to develop a range of middleware solutions (i.e. algorithms, protocols, services) for resilience to be applied in the design of highly available, reliable, and trustworthy networking solutions*”.

The main objective of this deliverable is to complement the work presented in deliverable D2.3 (Service level resilience solutions for the ad hoc domain) [11], focusing on ideas that typically correspond to improvements, or would allow improving some of the services presented in D2.3. These ideas either exploit the possibility of operating in an integrated way with the infrastructure domain, in addition to the exclusive operation over ad hoc environments or, more generally, exploit trade-offs that are relevant when operating on the ad hoc as well on the infrastructure domain.

Nevertheless, for completeness reasons, and in order to provide to the reader an outlook of the fundamental building blocks of HIDENETS, the deliverable also contains a general introduction that includes parts of text also included in deliverable D2.3 and in previous deliverables, as well.

## **Contents of the deliverable and relation to other work packages**

The work in WP2 was structured in three main tasks. The first one (Task 2.1) was mostly related to architectural issues, and the remaining two (Tasks 2.2 and 2.3) were concerned with service aspects. The first two deliverables in this work package (D2.1.1 and D2.1.2) presented results concerned with all the tasks, besides presenting the proposed architecture. The task concerned with architectural issues has been finished by the end of month 24. The present deliverable is exclusively focusing on research work related to the middleware services. The main architectural aspects of HIDENETS, including the general system architecture and the node software architecture, are briefly reviewed in the introductory section, which is the first section of this deliverable.

Section 3 addresses the problem of fault detection from the perspective of end-to-end services, which is studied under a probabilistic scope and which is suitable for IP based communication systems in general. Then, Section 4 provides a study on the dependability/performance trade off for replicated servers in the infrastructure domain. In Section 5, an extension to the Intrusion Tolerant Agreement service is introduced, which exploits the assumed availability of a reliable server in the infrastructure domain to improve the performance of the basic (non-extended) service, as presented in D2.3.

Differently from deliverable D2.3, the work described in this deliverable is not focused on specific services, and does not include details of service descriptions, interfaces or implementations. This is justified by the fact that in this deliverable, the work is concerned with solutions for the infrastructure domain that have not been included in any of the proof-of-concept prototypes developed in WP6. Therefore, the work presented here is more concerned with providing a detailed discussion of research issues, problem descriptions and proposal of corresponding solutions, evaluation through simulation and discussion of challenges and benefits of the proposed solutions. Therefore, the ideas presented in this deliverable could serve as the basis for future developments or implementation of concrete middleware services (or improved versions of the already developed ones), but which we leave as future work.

The deliverable concludes with some remarks about what has been done and some perspectives for the future.

## 2 Middleware resilience solutions

In this section, we provide some context for the work described in the deliverable. We start by reviewing the reasons underlying the decision of considering different operation domains, namely the infrastructure and the ad-hoc domain, which justify the fact that we have two “twin” deliverables, the present one and deliverable D2.3 [11]. Then we provide:

- a brief introduction to the architectural approach adopted in HIDENETS, which has an impact on the way how services are organised,
- an overview of the work previously done in WP2, namely the node software architecture which includes the services described here
- and, finally, an introduction to the complex middleware services which are detailed in D2.3 and which are referenced in the context of this deliverable.

### 2.1 Operation domains in HIDENETS

In HIDENETS, the ad-hoc and the infrastructure domains have been distinguished from the beginning (see e.g. the technical annex in the project proposal). The differentiation is necessary because the two domains clearly have different characteristics. E.g., car-to-car communication scenarios require mostly ad-hoc communication support and may not have access to any infrastructure domain. From a dependability point of view, communication performed in an ad-hoc environment poses a number of hard requirements not present in the infrastructure domain (e.g. dynamic behaviour of communication end points).

Some of the most challenging characteristics of ad-hoc domains are the following:

- Unreliable communication due to the presence of wireless links;
- Uncertain operational conditions, due to operation with variable numbers of users or traffic flows and due to highly dynamic network topologies;
- Variable failure modes, ranging from simple accidental faults to malicious faults (attacks and intrusions), due to operation in shared and weakly controlled environments.

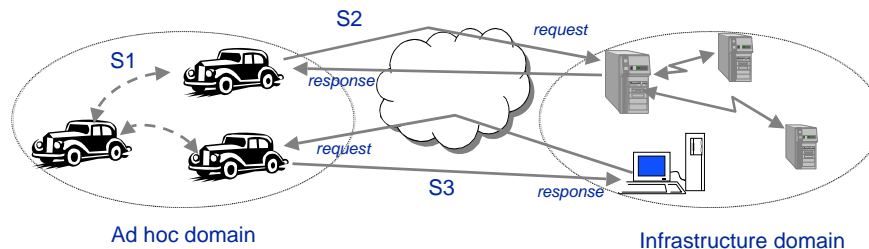
On the other hand, it is possible to consider scenarios in which additional infrastructure support is available, and this support might be useful to design improved solutions for integrated car-2-car communication settings. In fact, while in general the infrastructure domain consists of a back-bone IP network connecting both service providers and service clients, we can consider that parts of the ad-hoc domain may be connected to the infrastructure domain via various different means (e.g. WLAN hot-spots, IP over GSM, IP over 3G or 4G networks). In this way, this connection from clients in the ad-hoc domain to a server in the infrastructure domain can be exploited to improve performance and/or dependability, using a different range of solutions than the ones designed specifically for ad-hoc environments.

The different possibilities, or scenarios that have been initially identified, and which the HIDENTS project has proposed to investigate, are illustrated in Figure 1, and can be described as follows:

- S1) Both the service providing and the service accessing entities are located in the ad-hoc domain. Note that this includes scenarios in which the infrastructure domain is needed for connectivity, when the entities may not be within ad-hoc connectivity of each other. Examples of scenario S1 are the Platooning application and the first part of the Distributed Black Box application (the data distribution in the ad-hoc domain, see [8]), which have been explored in HIDENETS. In these applications we can observe the exchange of static and dynamic data obtained from the host vehicle, the road and other vehicles that participate in the traffic;
- S2) The service accessing entities are located in the ad-hoc domain and the service providing entities are in the infrastructure domain. Typically, this is the case for infotainment services which are accessed through the Internet, and also for emergency call scenarios. One particular case that we use in HIDENETS is once again the Platooning application, which may require all members to reach agreement on a platooning speed, and this agreement can be achieved with the help of a server in the infrastructure domain, if present and available;

- S3) The service accessing entities are in the infrastructure domain and the service providing entities are in the ad-hoc domain. This scenario is covered by the Distributed Black Box application example, in which collaborative backup of critical data generated at the vehicular nodes has to be done and therefore requires vehicles to act as servers of collected data to a client that is in the infrastructure domain and collects this data.

Scenarios where both service providing and service accessing entities are located in the infrastructure domain have been explicitly left out of HIDENETS concerns.



**Figure 1: Three service usage classes of scenarios.**

In terms of work structure, in WP2 we decided to distinguish between work on solutions possibly involving the infrastructure domain, and solutions focused on the ad-hoc domain. This is reflected in the existence of two tasks (task 2.2 and task 2.3) and two deliverables (the present one and D2.3). Most of the work was done having in mind the challenging characteristics of an ad-hoc environment. The middleware services developed in HIDENETS are included in deliverable D2.3 [11] as part of the resilience solutions for the ad-hoc domain. Here in D2.2 we provide complementary work that is relying on the existence of an infrastructure domain.

## 2.2 Architectural approach

Differently from previous approaches, in HIDENETS we follow a hybrid distributed system model approach, in which different parts of the system have different sets of properties and can rely on different sets of assumptions with respect to faults and synchronism. This has a number of advantages when compared to approaches based on homogeneous models, as explained in D2.1.2 [9] and in Reference [39].

One example of a hybrid distributed system model is the *Wormholes model*, named in this way after the astrophysics theory. This theory argues that one could take shortcuts, through, say, another dimension, and re-emerge safely at the desired point, apparently much faster than what is allowed by the speed of light. Those shortcuts received the inspiring name of *Wormholes*. In essence, Wormholes prefigure an ancillary theory which coexists with the classical theory, making possible subsystems which present exceptional properties allowing overcoming fundamental limitations of the systems under the classical theory.

The wormholes concept can in fact be instantiated in different ways. For example, when applied in the security domain, a wormhole takes the form of a security kernel, or a trusted component, as described in [13]. On the other hand, when timeliness is the relevant non-functional property to secure, the wormhole should essentially be timely. There are also examples of wormholes in the time domain. For instance, a very simple example of a timeliness wormhole can be a watchdog, which is able to shutdown a system or perform some other real-time action when a time bound is not met. Another example is a Timely Computing Base (TCB), as described in [40]. A system with a TCB has a control part, with synchronous properties, and a payload part, possibly asynchronous, in which the applications execute using a number of services provided by the former. A TCB constitutes an example of a timeliness wormhole.

The idea of an architecture based on the wormhole concept has been applied in HIDENETS. Therefore, a HIDENETS system can be subdivided in two parts, one “simple and trusted” and one “complex”. Each part is characterised by its own system and fault model, being one of those typically stronger than the other (e.g., synchronous with a crash fault model, which is stronger than asynchronous with arbitrary faults of processes

and communications). The stronger properties are secured by construction, i.e., the system is built in a way to make these better properties become effective.

In summary, a simplified perspective of the HIDENETS node architecture is presented in Figure 2. The highlighted block corresponds to a separate realm of operation and represents the more timely and trusted part of the system. It is also possible to see that this realm of operation can be built over a separate hardware infra-structure. On the upper part of the architecture reside the complex resilience middleware services, that used the optimised network protocols and/or the trusted resilience services. The HIDENETS services can be accessed by applications through specific HIDENETS interfaces, while SAF interfaces could be used to access other availability services. A detailed description of this simplified node architecture figure can be found in D2.1.2 [9]. Here we want to point out that the solutions introduced in this deliverable are intended to be used in the general part of the system to improve some of the *Complex Resilience Middleware Services*. On the other hand, the description of the implementation aspects of the *Simple and Trusted Resilience Services*, which we also call *Timeliness and Trustworthiness Oracles* (the term ‘oracle’ is used here to convey the idea of a better service, which does not fails), is provided in deliverable D3.3 [15]. This is because these are services related to the provision of (possibly strict) timeliness and security properties, which is an issue covered in WP3.

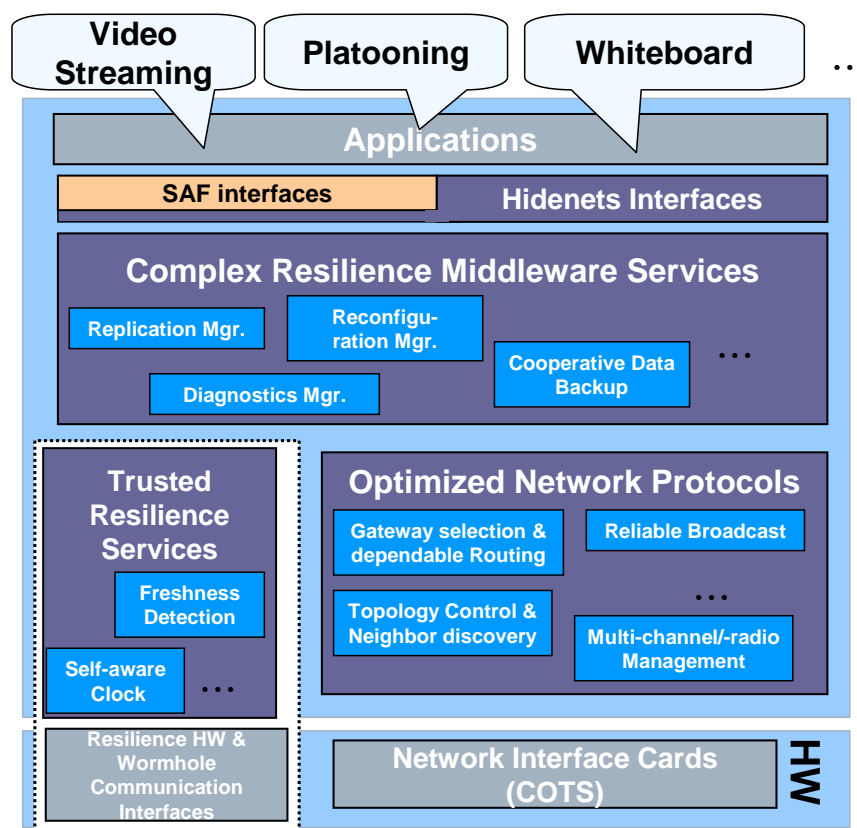


Figure 2: Simplified node architecture with Trusted Resilience Services.

## 2.3 HIDENETS node software architecture

The purpose of the node software architecture is to provide a coherent and structured framework where:

- i) the functional building blocks of a HIDENETS node are specified;
- ii) the relations between different building blocks are displayed;
- iii) the dependability related functions of interest to the supported applications are identified and made available through appropriate building blocks.

Figure 3 essentially illustrates the several building blocks, or services, that may be included in a HIDENETS node, while providing some additional information regarding the relationship of these building blocks within the node. This represents the HIDENETS architecture.

The layered structure as shown in the figure follows the standard ISO communication layer architecture. From the bottom, physical or hardware components are depicted including communication related components and other hardware that is relevant in the context of HIDENETS and that may be used in communication activities. This would correspond to layer 1 of the ISO standard. Then, several functional building blocks are depicted in layers 2 to 4, which offer typical network related functionalities. Some other services, like the In-Stack Monitoring and Error Detection, include functionalities that span across multiple layers. All these functional blocks are being addressed in the context of WP3 (as communication layer dependability improvements). Please note that some of the building blocks are represented with dashed lines. This means that although these blocks are considered necessary and are used by other services, they are not explicitly addressed in HIDENETS. That is, in HIDENETS we assume that existing implementations of these services are available and are adequate for the purposes of the project, and no particular improvement, modification or specification relative to these services is done in HIDENETS. This is the case, for instance, for the Session Control service, the Naming service, and the Group Communication service.

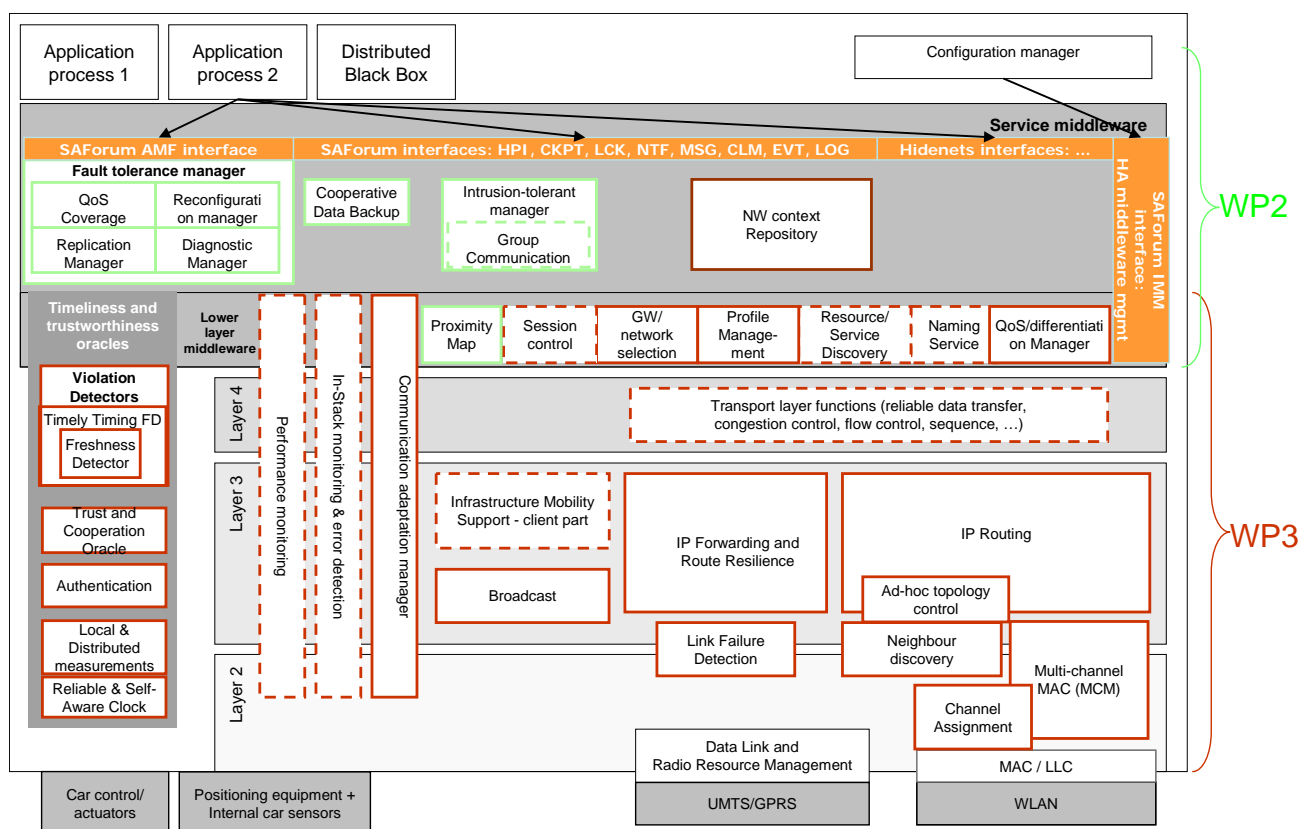


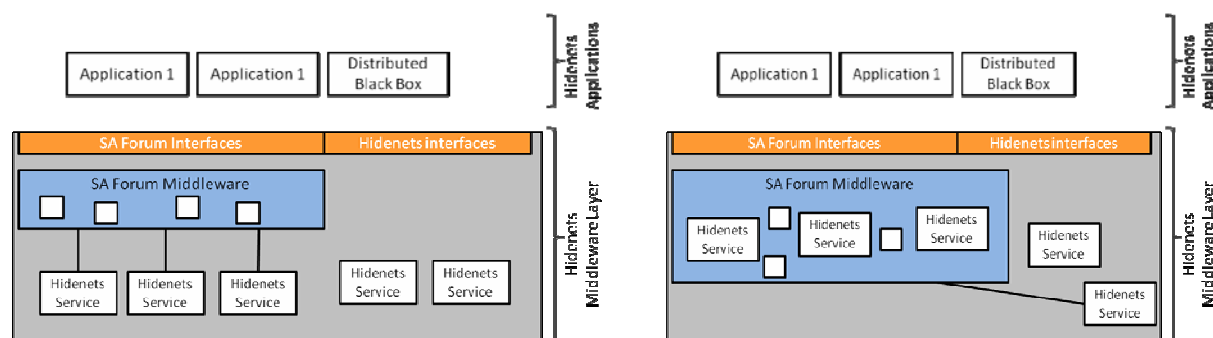
Figure 3: Overall picture of HIDENETS node software architecture.

On the left side of the figure, timeliness and trustworthiness oracles are grouped together. This separation is based on architectural principles established by the Wormholes model. Due to the nature of timeliness and trustworthiness oracles, which are supposed to provide strict timeliness and trustworthiness properties, the work concerned with their development and implementation was done essentially within task 3.3 of WP3. This is why they are not included in this deliverable, but are described in deliverable D3.3 [15]. We must note, however, that the initial work concerning the definition and design of these timeliness and trustworthiness oracles was done in the scope of WP2 and presented in the previous WP2 deliverables. We also note that time- or security-related functionalities may also exist outside the timeliness and trustworthiness oracles. E.g., applications may use standard system services like clock or authentication services. But since these services are not specifically related to dependability, we do not include them in this figure.

On the upper layers of the architecture, above the communication services, we consider the existence of a lower middleware layer which is composed of building blocks that are mainly targeted at interactions between middleware services and communication services. All these services are addressed in the scope of WP3. The remaining building blocks, which can be considered or named as middleware services, are supposed to handle higher level interactions with the applications. These middleware services are essentially addressed and developed in the context of WP2. The several blocks are coloured in red (darker) and green (lighter) to reflect the work package in which they are mainly addressed.

For certain uses of applications and services, it may be necessary that an application fulfils specific requirements, e.g. performance related requirements, which translate into requirements for middleware and communication services. Such requirements are registered through high level middleware services.

In Figure 3 the application interfaces are also represented, including the standard interfaces defined by the Service Availability Forum (SA Forum) and the interfaces that have been defined by HIDENETS.



**Figure 4: Relations of the SA Forum Middleware towards applications and HIDENETS services. Left figure: Middleware adapted to use HIDENETS services. Right figure: HIDENETS services integrated into the SA Forum Middleware.**

The SA Forum compliant service interfaces are provided by an SA Forum middleware implementation. This middleware can be either an implementation created from scratch, built intentionally on the HIDENETS architecture, or an existing implementation, modified to use the HIDENETS services for the internal operation. Figure 4 shows the difference between the two approaches.

When an existing middleware implementation is adapted to HIDENETS, then the existing building blocks, e.g. the communication subsystems, are modified to use HIDENETS services (see left figure). In this approach, the middleware provides only SA Forum services to the applications.

In the other case, the middleware is implemented from the ground up by using HIDENETS services. This allows the middleware to provide even those services besides the SA Forum ones and become an integral part of the overall architecture (see right figure).

The HIDENETS applications sit on top of this architecture and use the SA Forum and the established HIDENETS interfaces to access the provided services.

## 2.4 Complex resilience services

The Complex Resilience Middleware Services presented in Figure 3 are considered useful in the context of HIDENETS, in particular in the context of the ad-hoc domain. These services are presented in detail in deliverable D2.3 [11]. Here, we provide a very brief description of the purpose of these services, in particular as some of them are mentioned in the remaining sections of the deliverable. As solutions for the Diagnostic Manager can be applied both for the ad-hoc domain and the infrastructure domain, such a solution is also presented in this document.

Complex resilience services are developed on top of a potentially asynchronous model. The idea is that they will use and will rely on the trusted resilience services if and when required, mainly for the execution of critical steps in their operation. The following complex resilience services have been considered in the scope of WP2:

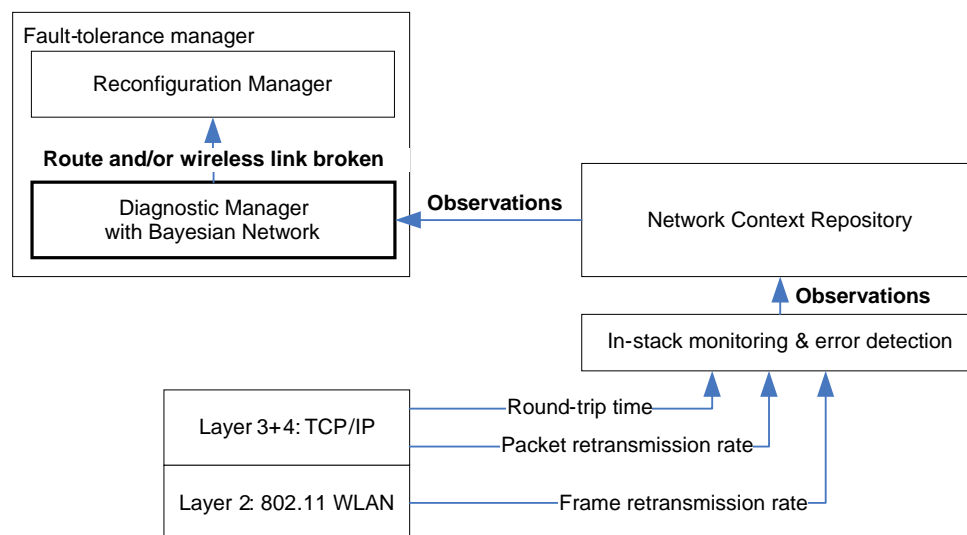
- *Diagnostic Manager.* The Diagnostic Manager is in charge of monitoring critical components of the middleware and of lower levels in order to judge the operational quality. E.g.: the Diagnostic Manager maintains information about the state of available/actually used network channels (ad-hoc or infrastructure); other middleware services or even applications can obtain this information in order to switch between different kinds of communication channels.
- *QoS Coverage Manager.* The QoS Coverage Manager interacts with the applications using information possibly collected by the Diagnostic Manager. It has to assess if the application requirements are satisfied or not. E.g.: when the available QoS changes in run-time, the service calculates the new operational parameters that should be used or assumed by the application in order to keep a constant coverage of the requirement.
- *Reconfiguration Manager.* The Reconfiguration Manager handles reconfiguration procedures. The reconfiguration could be static (in presence of a particular class of faults the Reconfiguration Manager chooses a predefined strategy) or dynamic (many strategies can be defined; the one used is chosen basing on the best predicted impacts on the dependability of the application).
- *Replication Manager.* The Replication Manager handles the state sharing of stateful services provided by nodes potentially both in the ad-hoc domain and in the infrastructure domain. The manager provides an interface for service replication. The Replication Manager selects replicas based on estimates of network performance and based on fairness policies given at service deployment time.
- *Cooperative Data Backup.* The Cooperative Data Backup is in charge of managing backups of critical data despite failures of the data owner and failures of the nodes storing the critical data for the data owner. This service is responsible both for dissemination and for recovery of data.
- *Proximity Map.* The Proximity Map provides information on neighbouring nodes.
- *Intrusion Tolerant Agreement.* The Intrusion Tolerant Agreement is used to reach agreement among a well-known set of participating entities, even when some of these entities might behave in a malicious way.

### 3 Probabilistic fault detection using cross-layer observations

#### 3.1 Introduction

As previously described, the Diagnostic Manager service monitors lower-level components and reports fault occurrences to higher level fault recovery components such as the Reconfiguration Manager. In that sense, the Diagnostic Manager detects and diagnoses erroneous components by observing component states either directly or indirectly by analysing output from relevant components. Based on the logic functions in the service, an optimal decision is made for recovering from the error before it propagates into system failures.

This section describes research concerned with enabling well-performing diagnosis in the Diagnostic Manager based on unreliable observations. The research is focused around a fault-detection and –diagnosis (FDD) framework employing a Bayesian Network (BN) to do the diagnosis. The integration of the framework into the Diagnostic Manager and the HIDENETS node architecture is illustrated in Figure 5. The detailed algorithms inside the framework as well as issues concerning the input observations and the output decisions are all described in the following sections.



**Figure 5: Bayesian Network (BN) fault-detection and -diagnosis framework integrated in HIDENETS Diagnostic Manager Service. Using round-trip time and retransmission rate observations, the BN detects and diagnoses underlying faults not observable.**

#### 3.2 Fault detection and –diagnosis framework

A required feature of the FDD component is the ability to operate on top of existing protocols. This is relevant to allow integration with existing and future IP based communication systems. Also, it should not require dedicated support functions in the network, but only use readily available observations from network traffic. It should be able to function across the infrastructure domain as well as in the ad-hoc domain.

These requirements make fault-diagnosis of end-to-end services a complex task due to multiple faults hidden in the network and due to the unreliable nature of observations that can be contradictory, ambiguous or simply missing [33]. Faults hidden in the network occur both in the ad-hoc domain and in the infrastructure domain. Yet, utilizing fault-tolerance upon an infrastructure network, fewer or more expensive recovery options may be available, which strengthens the importance of the correctness of the fault-diagnosis in the framework.

Considering the observation properties, the relations between multiple faults and observations are non-deterministic. A part of a solution is to introduce more observations from multiple layers to increase the amount of information. However, utilizing multiple observations for fault-diagnosis in mobile networks is non-trivial. As afore-mentioned, the individual observations are unreliable. Additionally, multiple

observations can also be contradictory due to different delays in the network traffic. The non-deterministic mapping between observations and faults implies the use of probabilistic methods for fault-diagnosis. In this section, we consider these issues and propose the use of Bayesian networks (BN) as a well structured probabilistic method to be applied in the detection and diagnosis process.

Based on cross-layer observations, a BN can take advantage of probabilistic and causal relations between faults, system behaviour and observations to estimate posterior probabilities of system states. An interesting strength of a BN is that it can handle missing, noisy, contradictory, and ambiguous observations. These properties are utilized to create a FDD component capable of detecting a fault that will lead to insufficient throughput and diagnose the cause as being either a bad link fault, congested link fault or both faults simultaneously.

In summary, the main contributions of this section are: (1) Development of a BN based on fault models, observations and their causal relations. The model is parameterized based on learning data synthesized from traffic observations. (2) The proposal of a FDD structure covering a strategy for the collection of data, processing of observations and calculation of posterior probabilities. (3) Proposal on how such an FDD structure can be integrated into the HIDENETS Diagnostic Manager service.

Existing research in network fault detection and diagnosis can be considered on different levels. A well studied diagnosis task is to determine whether a TCP packet loss is caused by congestion or a wireless link fault. In [20] HMMs are used and in [5] a Bayesian detector is applied. In both cases the approaches show how the probabilistic analysis can be used to improve TCP performance in mixed wired and wireless network environments. Moving to a cross-layer perspective the authors of [33] have incorporated BNs into fault localisation in complex communication networks with many nodes to handle scenarios when multiple alarms are generated. The essence of these results is that approximate inference methods can reduce the computational complexity of the FDD mechanisms with little impact on accuracy. In [33], also the robustness of BNs is emphasised in terms of noisy and missing observations as well as the occurrence of multiple concurrent faults. The authors in [18] have also deployed a BN for network state analysis based on network traffic. Yet this technique does not allow any fault diagnosis, and cannot directly be used for fault recovery.

### 3.3 Background on Bayesian networks

A BN is a graphical model that relates variables of a domain by their stochastic associations. The variables are discrete or continuous random variables and their relations are conditional probability distributions. Prior knowledge of the domain is represented in the structure of the variables, given by a graph, and in the strength of the relations between the variables from conditional probabilities. Based on knowledge of a part of the variables in the BN, inference about the state of the remaining variables can be performed. Via the inference process, a decision about one or more unobservable states can be made solely based on prior knowledge and observed variables.

Formally, a BN  $N = (G, P)$  consist of two basic entities [19]: a directed acyclic graph (DAG)  $G(X, E)$  where  $X$  is a set of nodes  $X = \{X_1, \dots, X_n\}$  and  $E$  is a set of edges connecting the nodes.  $P$  is a set of conditional probability distributions. Each node  $X_i$  represents a random variable with a finite set of values and each edge represents a causal relation between two variables. The distribution  $P(X) = P(X_1, \dots, X_n)$  represents the joint probability of all random variables in  $X$ . As the Markov property applies to nodes in the network we can define  $P(X) = \prod_{i=1}^n P(X_i | pa(X_i))$ , where  $pa(X_i)$  are the parents of  $X_i$ . Finally  $P = \{P(X_i | pa(X_i))\}$ .

This means that only a part of the conditional probabilities present in  $P(X)$  needs to be specified in  $P$ . These assumptions of independence make it practical to construct and parameterize such models. Most importantly, BNs also make inference in  $P(X)$  computationally feasible for models even with thousands of nodes and states [36]. More on the background of BNs may be found in [19].

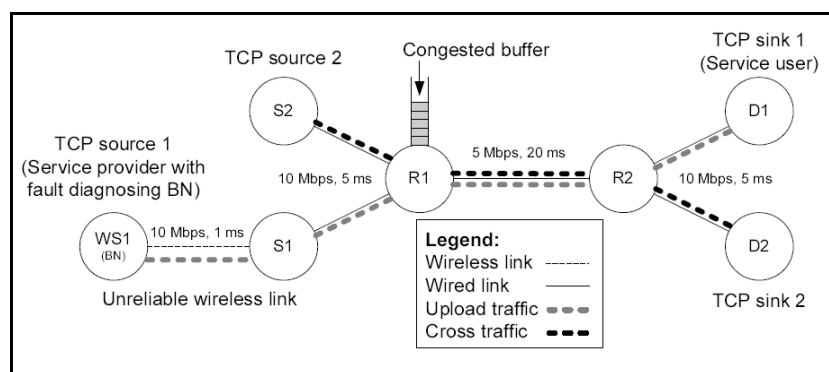
To construct a BN for inference  $G$  and  $P$  must be specified. Once specified, evidence  $e$ , which is assignment of a subset of  $X$ , can be propagated into the BN to infer states of hidden variables.

### 3.4 Bayesian network structure for the framework

Construction of a BN for fault-diagnosis is a three-step process: (1) obtaining domain knowledge, (2) developing a BN structure and (3) obtaining probabilities. The starting point for this process is a specification of the considered problem domain.

#### 3.4.1 Scenario specification

The scenario for fault-diagnosis defines a realistic set of faults and observations available to the BN. The scenario concerns the distributed black-box inspired application as described in [28]. The scenario contains a service on a node in a wireless network. Using a TCP connection it transfers data to a server in a wired network. Specifically a service failure is defined to occur when the throughput becomes insufficient for the transmission to succeed within a certain time frame. Thus, faults potentially leading to insufficient throughput are considered. A simplified model of this scenario is depicted in Figure 6 which will be used as a starting point for the subsequent fault specifications.



**Figure 6: Network model including a wireless link and a congested router.**

In this network model *TCP source 1* transmits data to *TCP sink 1* over a wireless link and a wired network with two routers. The wireless loss rate  $\pi_{\text{loss}}$  can be varied to simulate different link conditions. *TCP source 2* generates cross-traffic to induce congestion in the buffer of router *R1* controlled by a rate of Poisson process,  $\lambda_{\text{load}}$ , generating incoming connections. The incoming connections correspond to file transfers with sizes according to a Pareto distribution with an average of 10 KB and  $\beta = 1.5$ . This is a good approximation of Internet traffic [31].

#### 3.4.2 Fault specification

A cause of insufficient throughput is a high packet loss. TCP reacts strongly on packet loss as it infers congestion of the network route and, due to its congestion windowing mechanisms, reduces the transmission rate to maximise the throughput under the congested conditions. However, in cases where packet losses are due to a high frame loss rate on a poor wireless link the reduced transmission rate is undesirable. In both cases other recovery options could be e.g. to use another wireless link or change the destination of the transmission. However, to initiate the correct recovery, it is essential that the fault is detected and correctly diagnosed. The potential faults *congested route* and a *poor wireless link* are considered, because they lead to a high packet loss and a reduction in throughput. The congested route may be caused by sudden redirection of traffic due to a failed router. The poor wireless link is caused by the introduction of a noise source in the wireless frequency band. In both fault cases the repair rate is assumed to be low, i.e. the network fault remains within time requirements of the end-user service to transfer the file. From these assumptions the considered faults are permanent.

The two network model parameters  $\pi_{loss}$  and  $\lambda_{load}$  control the severity of the two faults. A normal state has been defined as a starting point. Next, the parameters have been tuned empirically in a simulation environment to cause a 50% throughput reduction. The resulting parameters are defined in Table 1.

Wireless loss [%]		$\frac{conn.}{Route Load [second]}$	
$\pi_{loss}$		$\lambda_{load}$	
Normal	High (Fault)	Normal	High (Fault)
1	4	18	47

**Table 1: Specification of the network model parameters defining normal and faulty states.**

### 3.4.3 Developing a BN structure from the scenario

To construct the BN structure  $G$ , variables and their causal relations are mapped from the problem domain to a graphical representation. To verify that the resulting model represents the problem domain, manual independence tests such as d-separation [26] can be used, but generally good patterns for construction and methods for verification are still open issues in the research domain of BNs [27]. The following structured method has been applied: (1) For BN fault diagnosis the first variables to consider represent the system components where faults may occur. (2) Next are the observable variables where useful information about the states of the unobservable system components can be obtained. (3) Finally, intermediate variables may be needed to describe system behaviour and relations between observations and the system components. (4) Edges are identified from causal relations between the system components and the observations potentially through intermediate nodes.

### 3.4.4 Fault nodes and observations

Essentially both the defined fault nodes can be modelled as two-state discrete random variables. I.e. Route,  $R = \{congested, not\ congested\}$  and Wireless Link,  $WL = \{poor, good\}$ .

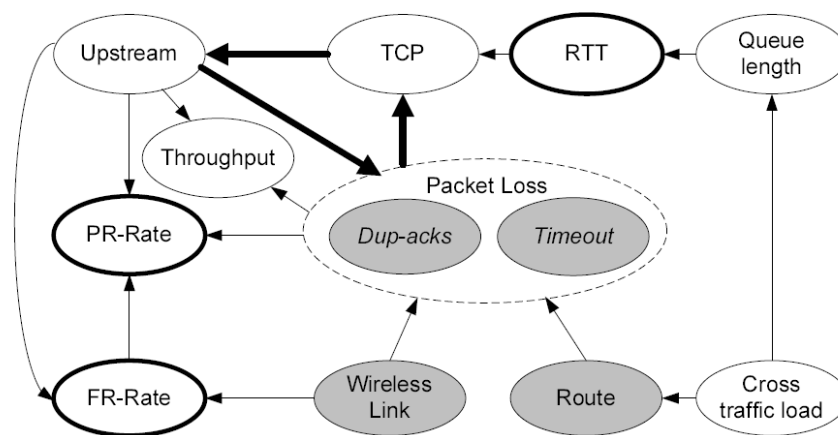
The criteria used in the selection of observations are that they must be causally influenced by either of the occurring faults and, at the same time, be available to the diagnosing BN from network traffic. The TCP endpoint of a connection over the wireless link offers several observation points. Measures such as packet round-trip time, packet retransmission rate, throughput and packet inter-arrival times are available from the transport layer. Likewise, radio signal strength indication, bit error rate and a frame retransmission rate are available from the link layer. These observations are all affected by the considered states of the transmission path. For instance, the round-trip time of a packet increases due to buffering when the channel is congested [5]. Likewise, the frame retransmission rate of the link layer increases when the quality of the link decreases [5]. The round-trip time (RTT), the packet retransmission rate and the frame retransmission rate are chosen as observation points of the network. These variables present adequate expressions of the influence of the faults to show the fault diagnostic capabilities of the BN and are typically readily available from the TCP layer and link protocols.

### 3.4.5 Intermediate model

The first model iteration is depicted in Figure 7. Greyed nodes are fault nodes (including packet loss), bold nodes are observation points and the rest are intermediate nodes. Notice that the two nodes Dup-acks and Timeout refer to individual packet loss nodes with individual edges. To simplify the figure they have been depicted with only one set of edges. The intermediate nodes are:

- **Cross traffic load** - The cross-traffic load is the load on the bottleneck router arising from other communicating parties.

- **Queue length** - The cross traffic load affects the length of the queue on the bottleneck router by arriving connections filling the queue with data. Included in this parameter is an assumption of a constant router service rate  $\mu$
- **Upstream** - The data-stream is modelled by *Upstream* that represents the actual amount of outgoing packets from the sender. It is assumed that there is always data to be sent from the application. The upstream is affected by the control mechanisms of TCP.
- **TCP** - The *TCP* variable covers the control mechanisms of TCP. TCP controls the upstream based on RTT and detection of packet loss. Notice that TCP will react differently depending on the type of packet loss experienced.
- **Throughput** - The rate of successfully acknowledged data.



**Figure 7: Intermediate model consisting of variables and causal relations.**

The fact that edges are missing between several nodes in the model should be interpreted as the variables being independent. Some idempotent independence assumptions are, for example:

**Upstream  $\rightarrow$  Cross traffic load** - It is assumed in this model, that  $Upstream = Cross\ traffic\ load$ . Thus, load on the bottleneck link is not significantly influenced by the upstream of the sending application. This also means that the influence of Upstream on Congestion is considered insignificant.

**Wireless link  $\rightarrow$  Route** - The two components of the network have no (significant) influence on each other, thus the nodes are regarded as being independent.

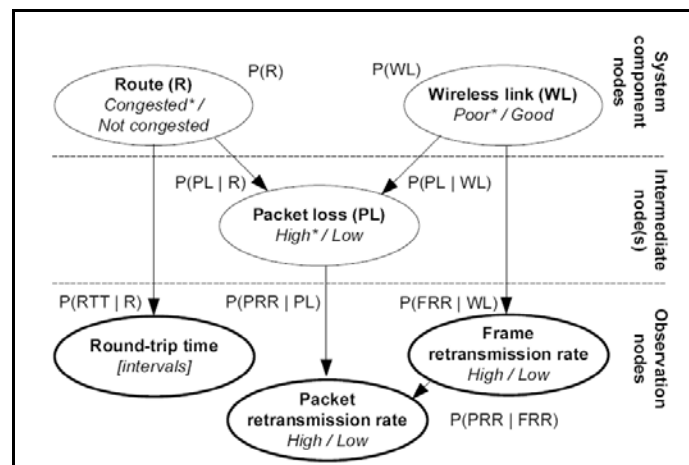
The model in Figure 7, however, is not a BN. It clearly contains a cycle  $Upstream \rightarrow Packet\ loss \rightarrow TCP \rightarrow Upstream$ . For this type of problem a dynamic BN could be introduced (see [24]). However, the attempt is to maintain the static BN. Thus the cycle must be eliminated. Moreover, some of the intermediate nodes may be disregarded. A few of these steps are emphasised subsequently:

1. The *TCP*-node is removed to eliminate the cycle in the graph. This clearly leads to a significant lack in the model. As *TCP* is removed, the two *Packet loss*-nodes are joined as the difference in packet loss is not considered.
2. The queue-length-node is of no direct interest other than as representing the load-state of the network. Thus it may be marginalised out of the *Cross traffic load*, i.e. calculated into all states of the *Cross traffic load* as a constant factor.
3. The state of the route is directly defined by the *Cross traffic load* node. Thus, these two nodes may be joined.
4. The remaining upstream-node can be eliminated from the model by converting PR and FR rates into ratios (packet retransmission ratio (PRR) and frame retransmission ratio (FRR)). Thus, upstream becomes a part of these observations.
5. Maintaining throughput as an observation has been considered an option. However, as the given throughput is strongly affected by the dynamics of the windowing mechanisms of TCP, the lack of TCP

in the model makes it difficult to use throughput as an observation.

6. To maintain the relation that both congestion and a poor wireless link lead to packet loss, the intermediate variable representing packet loss remains in the model.

The final outcome is the BN of Figure 8 named the basic model. In the figure the identified system components and observations are depicted as discrete variables along with their state-spaces in italics. States marked with an asterisk are the identified fault-states to be detected. As no knowledge of features in observations is available in advance, simple state spaces have been defined. The RTT is represented by a set of intervals representing the thresholds between discrete states. The PRR and the FRR have defined states as *high* and *low*, which are divided by a single threshold. Finally, the states of the *packet loss* variable are *high* and *low*. These states are not specified in details, as the *packet loss* node is an intermediate hidden node that is not observed.



**Figure 8: The BN structure containing both system components that may fail and observation nodes as well as causal relations between them, represented by edges and probabilities.**

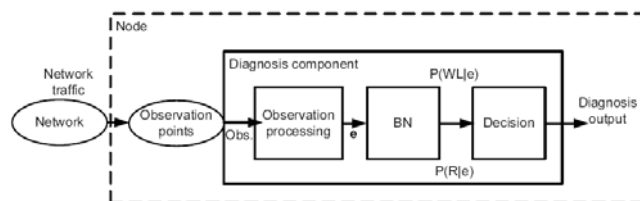
### 3.4.6 Observation processing

Bayesian Networks are not inherently capable of handling causality in (indefinite) time. Instead additional functions must be introduced to collect features from observation points to create evidence  $e$  for observations. A sampling approach is adopted where features are collected in discrete steps and observations are evaluated in that step.

The Round Trip Time (RTT) observation represents network delay and is extracted from incoming packet acknowledgements. The retransmission rates (Packet Retransmission Rate (PRR) and Frame Retransmission Rate (FRR)) are calculated based on retransmissions and transmission attempts. For these observation estimates, moving average mean estimators have been used. A mean estimator averages the passed observations in a time window. In this way, an RTT observation may be missing if no acknowledgments arrive within the time window.

### 3.4.7 Bayesian Networks used for fault detection and diagnosis

To use learning and to perform fault-diagnosis, data is needed. A framework has been developed that contains functionality for supplying observation data, both for parameterization and for fault-diagnosis. The framework is illustrated in Figure 9.



**Figure 9: The overall architecture of the fault diagnosis framework.**

Initially, data from the observation points is needed, which can be generated by a simulation, read from a network log file, or monitored real-time from a communication process. Next, the data needs processing to fit in the observation variables in the BN. The observation points are continuous random processes, which are mapped into the discrete state observation variables in the BN. This processing and mapping is described in Section 3.5.

After observation processing, the evidence is propagated in the BN and inference is performed. Inferring with the BN is done by calculating the joint probability of the variables in the graph. Calculating the marginal probability of a single variable  $R$  in  $P(R|e)$  from the joint probability produces the posterior probabilities of the states of  $R$ . Such posterior probabilities enable decision on the state a variable is in. In fault-diagnosis, the probabilities lead to a decision about the occurrence of a fault in  $R$ . Although other threshold based criteria can be employed, we assume here that a fault is detected if  $P(R=fault-state|e) < P(R=normal-state|e)$ . The process of exact inference in BNs is NP-hard [36] and several approximate inference methods have been suggested [32]. However, as it is considered outside the scope of this work to regard computational requirements to the inference process, exact inference is applied. Exact inference means calculating the entire joint posterior probability distribution of the BN based on all the evidence. This process is NP-hard in the number of variables in the BN.

### 3.5 Parameterization of the FDD component

Having defined the BN structure and the method and framework for obtaining probabilities, this section describes the final steps for parameterization of the BN and the mean estimators. The FDD component has been realized in a simulation environment (using ns-2 [22]) that implements the network model of Figure 6.

#### 3.5.1 Window size and sampling rate

The size of the window for the mean estimators is set in milliseconds individually for each observation as seen in Table 2. Empirical analysis has shown that the values in the table are feasible as initial setup.

The choice of sampling time is determined by the dynamics of the observations. In the used scenario, the dynamics of the observations are relatively slow due to the use of moving averages. Hence, a fast sampling strategy would not provide more information from the observations. Conversely, if the sampling rate is too low, it will have a negative impact on the reactivity. Sampling is done with a sampling period of  $T = 100\text{ ms}$ . From early simulation and test results, this setting seems reasonable to capture the dynamics of RTT and retransmission rates with varying window sizes.

RTT	States	10
	Thresholds	62-105 ms, 4.3 ms intervals
	Window size	300 ms
PRR	States	2 (Low, High)
	Threshold	0.03
	Window size	500 ms
FRR	States	2 (Low, High)
	Threshold	0.03
	Window size	500 ms
R	P(R=Not congested)	0.5
	P(R=Congested)	0.5
WL	P(WL=Good)	0.5
	P(WL=Poor)	0.5

**Table 2: Basic setup of the parameters and nodes for the BN.**

### 3.5.2 Obtaining probability distributions

The conditional probabilities in the BN can be elicited in several ways. One is to use domain knowledge from experts. However,  $P(RTT = t | route = congested)$  is difficult to perceive as it is very dependent on the properties of the given network. Therefore, a more useful method to elicit probabilities is based on learning. By using the Expectation-Maximization (EM) algorithm it is possible to estimate the parameters of a probabilistic model [17]. In the case of a BN, the parameters are the specific probabilities of the causal relations, e.g.  $P(RTT | route)$ .

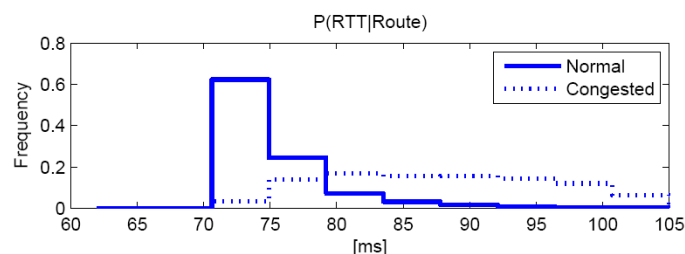
Two approaches can be used when learning with EM-algorithm: online and offline learning [24]. Offline learning uses pre-recorded sets of data to estimate the parameters of the model. When the BN is used for inference, no alteration of the parameters occurs. In online learning, the parameters are continuously estimated when the BN is in operation. This makes it possible to adapt the model to local network conditions.

Both approaches to learning have challenges; offline learning may require exhaustive data and online learning may not work with rare events. Solutions to some of these challenges have been suggested in [18].

Data sets for learning have been generated from the simulation containing information from both observation points and network states. For the fault-diagnosis process only observation information is generated. The probabilities of the network states, i.e. route and wireless link, are not conditioned on any variables in the BN. In this sense, they represent the prior belief of a fault occurrence in the network. In this setup, they have been specified equally as 50%/50% for a normal state or a fault. Due to their unconditioned properties, these prior probabilities can be used to adjust the sensitivity of the BN for fault-diagnosis. The final model configuration parameters are shown in Table 2. Learned conditional probabilities are listed in Table 3.

	$P(PL   R, WL)$		$P(PRR   FRR, PL)$	
	Low	High	Low	High
0, 0	0.927	0.073	0.968	0.032
1, 0	0.634	0.366	0.445	0.555
0, 1	0.345	0.655	0.178	0.822
1, 1	0.134	0.866	0.069	0.931
	$P(FRR   WL)$			
	Low	High		
0	0.845	0.155	0 $\rightarrow$ normal/low	
1	0.413	0.587	1 $\rightarrow$ fault/high	

**Table 3: Specification of the network model parameters defining normal and faulty states.**



**Figure 10: Probability distributions of RTT conditioned on Route state.**

### 3.6 The Bayesian Network in the HIDENETS Diagnostics Manager

This section describes how the FDD framework is integrated into the Diagnostic Manager (DM) service by relating interfaces to the node architecture of HIDENETS [9].

#### 3.6.1 Integration of the FDD framework

For the FDD framework to be integrated into a HIDENETS service, the input and output interfaces must be determined. The feasibility of the framework depends directly on the observations available for input and the possible recovery mechanisms which are dependants on the verdict from the DM. The basis for integration in the HIDENETS architecture is illustrated in Figure 5.

Recovery mechanisms:

- Reconfiguration Manager. DM supplies decision on fault-detection, and a diagnosis on whether the fault is in the route or on the wireless link. The assumption is that the Reconfiguration Manager (RM) is capable of recovering from either fault using the diagnosis.
- Low-level recovery mechanisms. Instead of going through the Reconfiguration Manager, the diagnosis consequences could be maintained directly by the low-level functions, such as networking and topology control communication components. These components would then have to have the ability to enforce a decision from DM independent of other communication functions.

### Input observations from network context repository / in-stack monitoring [9]:

- Round-trip time
- Packet retransmission rate
- Frame retransmission rate

All observations could be retrieved directly from in-stack monitoring.

### **3.6.2 Implementation**

Employing the BN, the DM can run periodically sampling the observations. As the BN is capable of handling one or more missing observations and still deliver a fault-diagnosis, a verdict on the fault status can be made at every sample. Once a fault is detected, the diagnosis is reported immediately to the external interface(s), to be utilized by for instance by the Reconfiguration Manager.

The sampling frequency of the mechanism can be set quite high (e.g. 1000 samples pr. second) for fast recovery possibilities, resulting in increased load from the inference engine at run-time, as more samples must be processed. Crude measurements show that a failure estimation based on a sample takes approx. 0.5 ms to generate, which must be considered when setting the sample frequency.

Several possibilities for inference algorithms inside the FDD exist. On one hand, exact algorithms guarantee the optimal diagnosis available in the model, but the execution time to reach an exact solution is unbounded. On the other hand, approximation algorithms can guarantee delivery of a diagnosis within a bounded time. In the latter case, the confidence in the presented diagnosis may consequently be lower than in the diagnosis of the exact methods.

## **4 Optimised dependability/performance trade-off for replicated servers in the infrastructure domain**

### **4.1 Introduction**

Server replication improves service dependability but it also adds overhead and complexity, which may worsen the performance. It is therefore essential to investigate how to minimize the performance degradation introduced by replicating servers, while maintaining the improved dependability levels allowed by server replication. Two characteristics of replicated systems can be improved in order to optimise their performance: failure detection and recovery triggers.

In deliverable 4.2.2, to be provided by the end of the project, the failure detection and failover settings are investigated in order to enhance the standard fault tolerance mechanisms implemented by distributed server replication platforms. In the analysis to be conducted and detailed in D4.2.2, the influence of several input parameters (or external parameters) and fault tolerance parameters (or internal parameters) is evaluated against three output metrics reflecting the trade-off between dependability and performance.

In this deliverable, we show how the simulation results can be used to select the optimal settings for the failure detection and recovery mechanisms in order to control the dependability/performance trade-off according to system/application requirements. The corresponding implementation of the selection and control logic is proposed for the HIDENETS node software architecture. This implementation mainly relies on the reconfiguration manager—in association with the replication manager—to select the optimal settings and infer them to the HIDENETS nodes where the failure detection and recovery processes are running (i.e. the failure detector and the client nodes).

### **4.2 Summary of the Simulation Work**

This section summarises some important aspects of the analysis/simulation work to be provided in D4.2.2. For more details about the model, results and conclusions refer to the latter document.

#### **4.2.1 Architecture Overview**

In [REF D4.2.2], the dependability/performance trade-off for replicated servers in the infrastructure domain is analyzed via simulations. The Stochastic Activity Network (SAN) formalism is used—with the Möbius modelling tool—to model client-server applications made dependable by implementing server replication. In the scenario investigated, the specific application is Session Initiation Protocol [29], which provides session management in IMS (IP-Multimedia Subsystem) environments and the distributed replication framework is the Reliable Server Pooling (RSerPool, [1][34][35]).

The IMS provides control functions and service enablers for IP-based multimedia sessions, which is relevant in the HIDENETS context because:

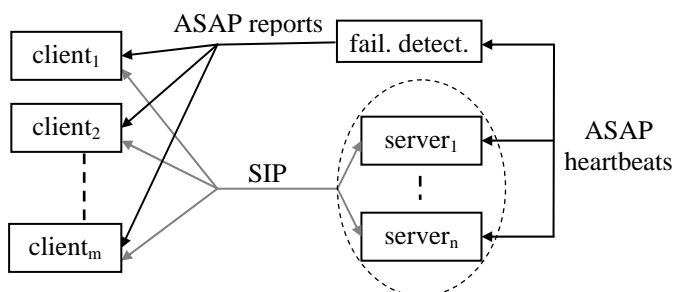
- it can support IP-based multimedia services—such as VoIP, video conference, instant messages, push-to-talk, online games and auctions—to users in their cars or in public transportations;
- it provides the means to control (via negotiation) the QoS resources allocated to a given session, which is a crucial feature to apply the decisions made by HIDENETS services (e.g. QoS coverage & reconfiguration manager) in an IP-based environment.

HIDENETS servers deployed in the infrastructure domain, e.g. emergency centres along the freeway, would greatly benefit from the integration of the RSerPool platform in terms of dependability. The RSerPool protocol suite manages the replicated servers and implements failure detection and failover functions that aim at masking communication faults (packet losses and long communication delays) and server faults (temporary crashes). The most important RSerPool protocol in this work is the Aggregate Server Access Protocol (ASAP). ASAP is used by the failure detector to:

- check on the replicated servers' status—the failure detector regularly sends ASAP heartbeats to the servers;
- report the servers' status information to the clients—the latter use the server list contained in the

ASAP report received from the failure detector to make its decision about which server it should send the next request to.

Figure 12 shows the RSerPool architecture for the example of replicated IMS servers.



**Figure 11: An example of an RSerPool architecture applied to the IMS system.**

#### 4.2.2 Model Parameters

The input/external parameters are measures of some uncontrollable system characteristics that particularly condition the dependability level and its corresponding performance level:

- The network delays and processing times, and their distribution,
- The packet error rate (PER),
- The server availability ( $P_{ON}/P_{OFF}$ ), the ON and OFF periods durations (time to failure (TTF) and time to repair (TTR)) and their distribution,  
TTR is specifically defined here as the time interval between the moment the sever fails and the moment the server is available again.

Fault tolerance provided by server replication mainly relies on two mechanisms, namely failure detection based on a heartbeat monitor and recovery. Therefore, the controllable fault tolerance parameters can be divided into two categories:

- failure detection
  - heartbeat frequency,
  - heartbeat timeout,
  - server status report creation at the failure detector and its mode of dissemination to the clients,
  - application request timeout.
- recovery
  - size of the server set, i.e. number of replicated servers,
  - number of application request retransmissions per server,
  - number of server failovers,
  - server selection policy (SSP).

The three output metrics used for this work are:

- *dependability*, defined as the probability that any application request is successful – note that in this work, dependability and availability are synonymous;
- *service access time*, defined as the average duration of successful application transactions;
- *overhead*, defined as the average number of packets required for each application transaction, including the communications steps supporting the fault tolerance mechanisms such as the server failure detection.

### 4.2.3 Conclusions of the Simulation Work

The influence of input variables such as communication delays and parameters from the fault models was analyzed. Simulation results using the standard IMS model showed that:

- higher packet error rates (PER) raise service access time (SAT) and overhead, but hardly impact dependability;
- server faults cause transaction failures, i.e. lower dependability, and SAT and load decrease (both because of our specific SAT definition that only takes into account successful transactions);
- the server failure model (in particular the ratio between the average packet round trip time (RTT) and the time to repair (TTR)) is very important because it determines how likely retransmissions can mask server failures.

The analysis of individual fault tolerance mechanisms revealed that:

- Fault tolerance mechanisms improve dependability at a fixed overhead cost, which depends on the failure detection and recovery settings. To significantly improve dependability, the additional fault tolerance overhead (introduced by heartbeats and server set status reports between an independent failure detecting node and the clients) becomes much larger than the decrease of overhead due to application request retransmissions avoided thanks to the failure detection mechanism.
- Recovery and failure detection mechanisms rely on parameters that can be optimally tuned for a given fault model and set of dependability/performance requirements. For the replicated server scenario—and for the specific fault and traffic models, and the server selection policy tested—it was shown that:
  - when the number of failovers increases (and the number of retransmissions per server correspondingly decreases), both dependability and SAT improve but the overhead greatly increased;
  - the server set should not have more servers than necessary to execute the maximum number of failovers for the current recovery configuration—when the server set deploys extra servers, all output metrics get worse, especially for configurations with fewer failovers allowed;
  - the impact of the heartbeat frequency setting varies greatly for each recovery configuration so it is difficult to draw general conclusions about its influence on the trade-off. It is suggested that once the recovery configuration has been selected, a few heartbeat frequency settings should be tested in order to optimise the output metrics for the given input fault model;
  - finally, having an accurate input model of the communication delay distribution permits to ideally set the application request timeout, which significantly improves the trade-off, especially with recovery configurations favouring server failovers over numerous retransmissions per server.

### 4.3 Simulation Model Application

Building the simulation model helps understand and evaluate how server replication-based fault tolerance influences dependability and performance. Results obtained with the simulation environment can be used to validate analytical models that evaluate dependability and/or performance metrics in similar network/communications/fault scenarios.

More importantly in the HIDENETS context, the simulation environment can be used to determine which fault tolerance configuration would be most beneficial for a system most and at what cost.

In this subsection, several strategies for optimal configuration selection are analyzed and an example implementation of the configuration selection logic in the HIDENETS node software architecture is proposed.

### 4.3.1 Configuration Selection Criteria

Each fault tolerance configuration tested is evaluated against the three output metrics: dependability, service access time (SAT), and overhead. This makes it nearly impossible to determine which configuration is optimal by just looking at the results generated because it is very unlikely that a single configuration can return optimal values for the three output metrics simultaneously. Therefore, some criteria are necessary to guide the selection process.

#### 4.3.1.1 Output Metric Thresholds

The simplest way to rule out fault tolerance configurations during the selection process is to set a threshold value for each output metric. These thresholds should translate some of the system and/or application requirements. E.g. users might demand that dependability is above 99%; the QoS requirements from an end-user application (e.g. online auctions) impose that SAT is below 300ms; and a system could only support a maximum load of 15 packet units per transaction.

Note that this technique might still leave multiple configuration candidates for the final choice. If one of the metrics has a higher priority than the other two, such as dependability would be in safety-critical systems, only two output metrics are bounded and the configuration that gives the best level for the third metric is picked.

#### 4.3.1.2 Score Function

If no limit is imposed on less than two output metrics, a score function is needed in order to rank each fault tolerance configuration by returning a single score value for a given combination of output values. The score function is made up of contribution factors (CF)—usually one CF per output metric for which a threshold is not required. A contributing factor can simply be the ratio between the specific configuration output level and the standard output level. Note that for the dependability contributing factor definition it is more relevant to consider the ratio between undependability output levels obtained for a specific setting and the standard setting because:

- the relative difference between standard and replicated IMS dependability levels can be so small that the magnitude of this ratio is much lower than that of the SAT and load ratios and, thus, becomes insignificant in the score function;
- this way, the variations of each contributing factor represent the system behaviour similarly:  $CF_{undep.}$ ,  $CF_{SAT}$ ,  $CF_{load}$  increase/decrease when the system behaviour worsens/improves respectively.

When one threshold is defined, the score function is calculated for the subset of configurations that respect the threshold requirement.

In Table 4, some score function examples are given. The first example does not favour any output metric for the selection process but the next two functions are shaped so that  $CF_{undep.}$  variations have a bigger incidence on the final configuration rankings.

Score function	CF priority
$CF_{undep.} \cdot CF_{SAT} \cdot CF_{load}$	Fair
$\exp(CF_{undep.}) \cdot CF_{SAT} \cdot CF_{load}$	$CF_{undep.}$
$CF_{undep.} \cdot (CF_{SAT} + CF_{load})$	$CF_{undep.}$

**Table 4: Score functions examples.**

### 4.3.2 Configuration Selection Techniques

The optimal fault tolerance settings for the mechanisms of the replicated platform can be selected either (1) when the system where replication is implemented is being designed or (2) dynamically adapted to the current conditions of the system. The characteristics of both the design time and the run time modes are discussed next. Advantages and drawbacks of each approach are analyzed and the specific HIDENETS services and procedures that allow implementing each mode are presented.

#### 4.3.2.1 Design-Time Configuration Selection Technique

Simulations are run for a given set of input parameter settings that are drawn from system specifications and/or average values measured in a similar existing system. The output metrics are evaluated for the selected set of input settings and a range of controllable fault tolerance settings—heartbeat frequency, heartbeat timeout, number of replicated servers, number of retransmissions, number of failovers, application request timeout. The fault tolerance configuration that optimally meets the system/application requirements is selected using either technique introduced in the previous subsection and is implemented in the real system (cf. Section 4.4 for the specific HIDENETS implementation).

This approach becomes limited for systems whose fault models change over time. If the fault model varies with time, it is not ideal to pick the optimal configuration based on simulations using only one set of input fault model settings. One way to include this aspect in the offline process is to create aggregate input values. For instance, the overall PER (packet error rate) input value could be the sum of each PER level in a system times the individual fraction of time that the system experiences each PER level, as shown with the following equation. The danger with this approach is that the output metrics have not been proven to vary linearly with the input variables so the configuration for the average PER,  $P_{OFF}$  (server unavailability) and CL values might not be the one that gives optimal average dependability, SAT and load.

$$PER_{input} = \sum_i \Pr(PER = PER_i) \cdot PER_i$$

#### 4.3.2.2 Run-Time Configuration Selection Technique

In order to cope with dynamically changing fault models, the run time approach should be considered whenever possible. Run time fault-tolerance configuration tuning assumes:

- A database containing the results of the subset of configurations to be used for a given set of input variable values corresponding to the current system state.
- Real-time input metrics measurement techniques
- Protocol extensions in order to communicate the dynamic input values to the entities such as failure detector and clients that control the configuration parameters. E.g. the failure detector can adapt the reporting scheme (heartbeat timeout values, report frequency, report acknowledgements) according to the current PER and RTT levels for communications between the failure detector and servers; the clients can adapt the recovery strategy, e.g. by increasing the number of failovers when server failures are long or adapt the application request timeout according to RTT (client-server communications).

## 4.4 HIDENETS-specific Implementation

In order to exploit the simulation results, the different fault tolerance mechanisms which the optimization relies on should be implemented in the HIDENETS node software architecture.

### 4.4.1 Design-time Implementation

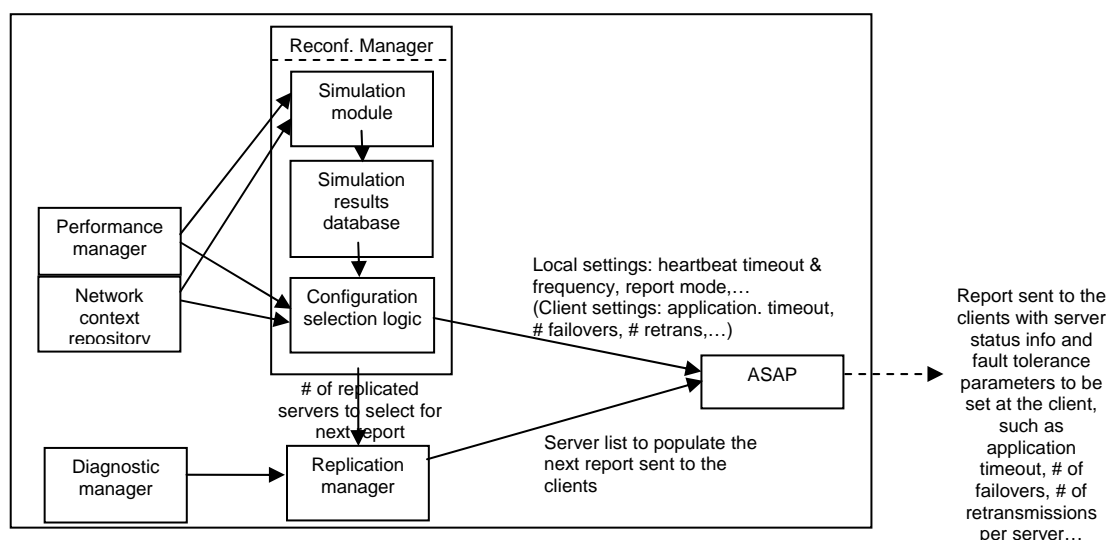
In design-time mode, the optimal configuration is selected with fixed input values and the configuration settings are ‘hard-coded’ in the respective fault tolerance functions:

- Heartbeat frequency and timeout (failure detector node)
- Application timeout (client nodes)
- Report mode (failure detector and client nodes)
- Number of retransmissions and failovers (client nodes)

#### 4.4.2 Run-time Implementation

For the run-time mode, we suggest that the overall configuration selection process is implemented as a sub-service of the Reconfiguration Manager in the failure detector node because this HIDENETS service is part of the Fault Tolerance Manager and it is responsible for triggering the necessary configuration changes to provide fault tolerance.

Figure 12 illustrates the failure detector node software architecture for the selection and implementation of the optimal fault tolerance configuration, as well as the interfaces between the HIDENETS services involved for the run-time solution.



**Figure 12: Failure detector node software architecture.**

- Simulations output values (and the corresponding input parameters and fault tolerance parameters) are saved in a database, which should be a part of the Fault Tolerance Manager as well—maybe directly in the Reconfiguration Manager if no other HIDENETS service uses the information saved in the database. Simulation results can also be generated online, running simulations in the simulation module of the failure detector node with input values obtained from the Performance Manager and the Network Context Repository.
- The configuration selection logic requires information about the current system characteristics, which are obtained from the Performance Manager and the Network Context Repository. The selection logic implements a selection strategy and picks the optimal configuration in the database using the current system characteristics (i.e. mapped to input parameters values).
- Heartbeat timeout and heartbeat frequency are managed by the RSerPool-specific ASAP layer. In the design time mode, these parameters can be (1) preset in the ASAP layer when the failure detector node is deployed in the system or (2) the fault tolerance configuration selection service in the failure detector requests the simulation results from the database, selects the optimal configuration, and infers the desired setting to the ASAP layer via a dedicated interface. In the run time mode, the latter option is mandatory as the optimal configuration is selected dynamically according to the fluctuations of the system behaviour.
- Before the failure detector sends the server set status information to the clients, a report needs to be created. To do so, the Replication Manager requests this information from the Diagnostic Manager. The latter is in charge of gathering all failure detection information it can obtain from the different functions implemented and processes it to determine the status of each server in the replicated set. The minimum input data to the Diagnostic Manager in our scenario is the heartbeat information at

the ASAP layer.

Note that in some implementations, the list with each server's status from the Diagnostic Manager could be intermediately post-processed by the Replication Manager, for instance to re-order the servers in the list according to the server selection policy. Finally, the information to be put in the reports to the clients is sent to the ASAP layer.

Note that in both design and run time modes, the report procedure is run using the same mechanism as reports are mandatory for accurate server selection at the clients.

- The report mode determines when the reports containing the server status list should be sent to the clients. Examples of report modes include 'regular' (the failure detector multicasts the report to all clients after every round of heartbeat) or 'on-request' (the failure detector sends the latest version of the report to an individual client when the latter requests the list). A specific report mode should be either:
  1. preset in the failure detector at deployment time
  2. or inferred by the failure detector or the clients when the clients first contact the failure detector (usually after connecting to the system) in order to obtain the list of replicated servers that implement the requested service. This information would be exchanged via the interface between replication manager and ASAP layer.

Note that in the analysis conducted in [REF D4.2.2], only the 'regular' setting was tested.

The parameters and functions implemented at the client nodes are:

- Application request timeout, maximum number of retransmissions per server, and maximum number of server failover parameters should all be set at the application layer in a similar manner as the heartbeat timeout and heartbeat frequency are set in the ASAP layer at the failure detector node. Therefore, a dedicated interface is required between the configuration selection service and the application layer.

For instance, the application could systematically ask the Replication Manager if it should retransmit a pending request, and to which server. This would require consulting the client-local Diagnostic Manager in order to optimise the probability to send the request to an available server.

## 4.5 Summary

In this work, a SAN model was designed in order to evaluate the dependability/performance trade-off for distributed replication platforms such as RSerPool, and for the example of IMS servers. The main contribution with respect to dependability is the possibility to control the dependability level in the system, in correlation with the performance level.

The model is abstracted enough to be applicable to most client/server applications deployed from the Internet and it can be used to test a wide range of system characteristic input values and fault tolerance settings. Consequently,

- the model can be reused with many systems and for many network/traffic/failure model scenarios;
- different fault tolerance mechanisms configurations can be tested so that the optimal configuration is selected and implemented in the system considered. Strategies for implementations of optimal configuration selection algorithms were discussed but no novel solution was devised.

Finally, the practical implementation of the optimal configuration selection (independently from the selection algorithm itself) in the HIDENETS software architecture was also proposed, and relies on multiple HIDENETS middleware services that interact with the RSerPool and application fault tolerance mechanisms:

- In design mode, results are pre-saved in the result database and the configuration selection is done ONCE at deployment time.
- In partial run-time mode: results are also pre-saved in the database BUT the configuration selection mechanism regularly (or event-driven) assesses the network characteristics and determines which fault tolerance configuration should be adapted.
- In full run-time mode, the simulation module is fed with input values from the Performance Manager and Network Context Repository and generates new results to be processed by the configuration selection mechanism.
- Whether the operation mode is design-time or run-time, reports are regularly sent by the ASAP layer of the failure detector node. The replicated server status provided by the Diagnostic Manager is filtered by the Replication Manager according to the number of servers to put in the report (this is decided by the configuration selection logic). Additional filtering criteria such as state consistency and current server memberships can also be implemented by the replication manager.
- The client software architecture is not the same as the failure detector software architecture. Mobile Client nodes should save as much battery/computation power as possible; therefore, we assume that all parameters are selected at the failure detector node and passed on to the clients in the ASAP reports. The parameters put in the reports are application timeout, number of retransmissions per server, number of failovers, report mode. If the clients have enough resources (e.g. when a laptop is connected to a power plug in the train) they could also implement their own parameter selection logic using input from the Network Context Repository, Performance Monitoring and Diagnostic Manager.

## **5 Infrastructure-assisted Intrusion Tolerant Agreement**

### **5.1 Introduction**

Consensus is an important problem in distributed systems. Many tasks, such as state-machine replication, atomic commitment, and total order multicast, can be reduced to consensus. In the past decades, many theoretical and practical algorithms have been proposed for a large variety of system models.

Practical solutions to the consensus problem play an important role for the construction of dependable networked embedded systems. For example, consensus can be used to coordinate the actions of distributed mobile actors, such as robots or cars. The properties of consensus guarantee that all correct participants make a common decision that cannot be disrupted by faulty entities.

The system environment of distributed embedded systems shows some differences to “traditional” distributed systems. Nodes have significant constraints on memory, CPU power, and communication capacity, and there may be a higher fluctuation of nodes. At the network level, many systems have hybrid communication facilities, composed of ad-hoc communication and infrastructure communication. The ad-hoc network allows direct communication between nodes within communication range. Sometimes, a multi-hop routing implementation allows decentralized communication between nodes on a larger scale on the basis of the ad-hoc network. In addition, access points provide connectivity to a static infrastructure.

Some recent research investigated consensus algorithms for mobile networks. These algorithms are either designed for autonomous operation using only an ad-hoc network, or are fully based on the support from a central infrastructure. In the ad-hoc network, the participants can fluctuate very fast. Nodes are easy to attack, as often there is no physical control over the participants. Fully decentralized algorithms must cope with these problems. Solutions that rely on the central server often are more efficient. However, a central server that is required for operation represents a single point of failure, a very undesirable design for dependable applications. Also, moving nodes must handle hand-over between access points; with current standard technology, these hand-overs may cause lack of connectivity to the infrastructure for durations up to several seconds.

Here we discuss some approaches to better optimise distributed consensus algorithms considering that entities operate in a mobile wireless ad-hoc network that allows direct communication between them, while road-side access points, where available, provide connectivity to a static network infrastructure. The main focus is on finding strategies that allow operation with and without connectivity to a central infrastructure. A system that supports both ways of operations can provide different liveness or timeliness guarantees, depending on whether there is connectivity to the infrastructure. For example, a central infrastructure can help to make stronger timing guarantees with high coverage on consensus termination. If the connection to the central infrastructure fails, a decentralized ad-hoc protocol could provide the same functionality with weaker timing guarantees.

In HIDENETS the availability of this service can be interesting for applications that i) require consensus to be achieved, despite the existence of possibly a subset of malicious participants, and that ii) may be able to communicate with servers in the infrastructure domain besides being able to communicate in the ad-hoc environment. The Platooning test bed, described in detail in Deliverable D6.2, is an example application in which this service might be useful.

## 5.2 Related Work

Several papers aim at providing solutions for consensus in mobile environments. Badache et al. [4] provide a multi-valued consensus algorithm for crash-stop failures in an asynchronous system with eventually strong ( $\diamond S$ ) failure detection. The authors assume that mobile hosts (MH) are connected to mobile support stations (MSS), and communication between MH is routed through MSSs (there is no direct ad-hoc communication between MHs). The main idea of the protocol is that each MSS acts as a representative of all MHs connected to it. Seba et al. [30] generalise this idea in a general framework for solving consensus in infra-structured wireless networks (i.e., wireless networks with connections to the infra-structure domain). This solution adds support for MSSs that dynamically join or leave the consensus protocol session due to hand-over of MHs.

Wu et al. [43] present a hierarchical consensus protocol for mobile ad hoc networks. This protocol works with crash-stop failures and assumes an asynchronous system with an unreliable failure detector of the eventually perfect ( $\diamond P$ ) class. It works fully decentralized without using any infrastructure. The algorithm introduces a hierarchy between at least  $f+1$  hosts that act as proxies, and the remaining hosts that are associated to a proxy. By merging messages in proxies, this approach reduces the total number of messages.

Another form of “hybrid” approach combines failure detection with randomization. Failure detectors and randomization are two ways of avoiding the famous FLP impossibility, which states the impossibility of consensus in asynchronous systems in the presence of crash failures [16]. Aguilera and Toueg [1] and Mostefaoui et al. [23] both propose a binary consensus algorithm of this hybrid kind for the crash-stop model. If the unreliable failure detector works correctly, deterministic termination is guaranteed; otherwise, the algorithm terminates eventually with probability 1.

All work discussed so far only uses either ad-hoc communication or infrastructure communication, but not both, and only covers crash-stop failures. Some authors have previously addressed Byzantine consensus for mobile networks. Angluin et al. [2] present a weaker form called “stabilizing consensus”, in which nodes do not commit to a final output at a certain point in time, but instead have outputs that eventually converge to a stable configuration. Drabkin et al. [14] discuss the related problem of efficient Byzantine broadcast in ad-hoc networks. Wang et al. [41] propose an algorithm for Byzantine consensus in mobile ad-hoc networks.

Here, we address the problem of Byzantine consensus. In open mobile ad-hoc networks, it is even easier than in traditional systems to inject malicious nodes. Thus, reaching consensus in spite of malicious nodes is an important mechanism. The main advantage of our hybrid approach is that it makes use of a hybrid network consisting of ad-hoc and infrastructure connections.

## 5.3 System Model

The system consists of a set of  $n$  processes  $P = \{p_0, \dots, p_{n-1}\}$  and an infrastructure service process  $p_s$ . The processes are said to be *correct* if they adhere to the protocol until termination. Otherwise, they are called *corrupt*. We assume a Byzantine fault model, i.e., there are no constraints on the actions of corrupt processes. Moreover, we assume that only  $f$  out of  $n$  processes can be corrupt, with  $n \geq 3f+1$ .

In the multi-valued consensus problem, each processor  $p_i$  proposes some initial value  $v_i$ , and then the processors must agree on a common value. As we assume a randomized model in an asynchronous system, the termination of the protocol can be guaranteed only in a probabilistic way. Corrupt processes must not be able to force the correct processes to do something “bad”. Formally, the consensus problem is specified by the following properties:

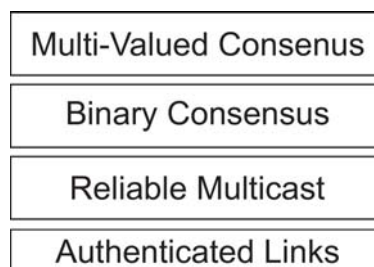
- **Validity 1:** If all correct processes propose the same value  $v$ , then any correct process that decides, decides  $v$ .
- **Validity 2:** If a correct process decides  $v$ , then  $v$  was proposed by a correct process, or  $v = \perp$ .
- **Agreement:** No two correct processes decide differently.
- **Termination:** All correct processes eventually decide with probability 1.

We further assume that the identity of all message senders can be verified. Without this assumption, an adversary could forge any message and consequently impersonate any node, and thus no solution would be

possible at all. In practice the verification can be assured with cryptographic message authentication mechanisms.

## 5.4 Design Options

Most binary and multi-valued consensus algorithms are constructed by a hierarchical composition. Figure 13 shows such a typical composition, in which multi-valued consensus is implemented on top of binary consensus and reliable multicast, with authenticated point-to-point links at the bottom.



**Figure 13: Hierarchical composition of multi-valued consensus.**

In the following, we distinguish three alternatives for creating an infrastructure-assisted solution, given an existing decentralized implementation of consensus. We focus on solutions that gain some advantage if the infrastructure connection is available, and otherwise work as a traditional decentralized protocol. Infrastructure interaction can be integrated in the existing building blocks at the level of reliable multicast, at the level of binary consensus, and at the level of multi-valued consensus.

### 5.4.1 Reliable Multicast

The level of *reliable multicast* implements a mechanism that guarantees that if a message  $m$  is delivered to a correct node, this (and only this) message is eventually delivered to all correct nodes. A typical algorithm that is able to tolerate malicious corrupt nodes is the reliable broadcast algorithm described by Bracha[7].

A reliable multicast might be supported efficiently by a central infrastructure node if this central node is guaranteed to fail only by crashing. The infrastructure could distribute the broadcast value, and only in the case of unavailability of the central node, the distributed participants would have to execute a distributed protocol. However, if malicious corruptions of the infrastructure node are included in the failure model, it becomes obvious that in any case, the set of nodes needs to execute a distributed algorithm to ensure the delivery of a unique value. We expect that an infrastructure exhibiting Byzantine behaviour cannot provide any significant benefit, and thus we do not discuss this option in more detail.

### 5.4.2 Binary Consensus

The level of *binary consensus* implements an algorithm that guarantees that all correct nodes agree on a common value, and if all correct nodes propose the same value, they decide on that value. Formally, the agreement and termination condition are the same as for multi-valued consensus as specified above, and the validity assumptions can be simplified to just **Validity 1**. Typical randomized binary consensus protocols are based on the random *coin-toss* operation, which returns the value 0 or 1 with equal probability. The coin tossing can be implemented with *local* or *shared* protocols. Typical examples of local coin-tossing are the algorithms of Ben-Or [6] and Bracha [7]. Shared coins are based on cryptographic distributed algorithms that provide the same random value to all participants. All the randomized algorithms work in rounds, and the complexity of the algorithm (in terms of time and messages) correlates to the number of rounds. Shared coin algorithms, such as ABBA [10], typically have an expected small, constant number of rounds, but require the execution of computationally complex cryptographic protocols. In embedded distributed systems, in which the participants typically have only small computation power, the computational complexity can render them impractical. On the other hand, local coin algorithms in general require an expected exponential number of rounds.

The coin tossing is the point where an infrastructure can provide essential support. A central infrastructure server can provide consistent random number to all participants. Let us first assume a crash-stop infrastructure. In this case, all nodes try to contact the infrastructure to obtain the random value. The

infrastructure reveals this value as soon as sufficiently many nodes have requested the value. This way, all nodes obtain the same random number, which results in the low expected number of rounds (as in existing shared coin algorithms), without the need of complex cryptography. If a node does not receive a timely response from the infrastructure, it creates a local random value, resulting in the behaviour of a normal local coin protocol.

```

Function MVConsensus( $est_i$ )
(1)  RBCast( INIT( $est_i$ ) );
(2)  wait until (valid COORD( $lv$ ) has been delivered or TIMEOUT expired);
(3)  if ( justified COORD( $lv$ ) has been delivered) then  $est_i \leftarrow lv$ ;
(4)  else wait until  $((n - f)$  INIT messages have been delivered);
(5)      if  $(\exists v : \#INIT(v) \geq f + 1)$  then  $est_i \leftarrow \langle v, justification \rangle$  else  $est_i \leftarrow \langle \perp, justification \rangle$ ;
(6)  RBCast( PRE( $est_i$ ) );
(7)  wait until  $((n - f)$  justified PRE( $x$ ) messages have been delivered);
(8)  if  $(\#PRE(x_1 \neq \perp) \geq f + 1, \#PRE(x_2) = 0 \text{ for } x_2 \notin (x_1, \perp))$  then  $b_i \leftarrow 1$  else  $b_i \leftarrow 0$ ;
(9)   $c_i \leftarrow$  BinaryConsensus( $b_i$ );
(10) if  $c_i = 0$  then return  $\perp$ ;
(11) wait until  $((f + 1)$  PRE( $\langle v \neq \perp, justification \rangle$ ) have been received);
(12) return  $v$ ;

Function Deliver (INIT( $v$ )) at Coordinator  $p_s$ 
(13) if  $((n - f)$  INIT( $v$ ) messages or  $f + 1$  identical INIT( $v$ ) messages have been delivered):
(14)   if  $(\exists v : \#INIT(v) \geq f + 1)$  then  $lv := \langle v, justification \rangle$  else  $lv := \langle \perp, justification \rangle$ 
(15)   RBCast( COORD( $lv$ ));

```

**Figure 14: Infrastructure-assisted consensus algorithm.**

The situation becomes more difficult with a corrupt infrastructure. In the worst case, the malicious infrastructure might distribute “bad” random values in a way that inhibits consensus termination. This problem can be mitigated in practice by limiting the infrastructure interaction to a small number of rounds. If the infrastructure behaves correctly, it is highly probable that the algorithm will terminate within this bound. Lack of termination within the limit is a strong indication of infrastructure corruption. The normal local-coin algorithm that is used after the bound will guarantee eventual termination with probability 1. The only adverse effect that a corrupted infrastructure can have is a delay of termination for a few rounds. In a system model in which the infrastructure is correct with high probability (but not guaranteed to be correct), this approach results in a low expected number of rounds.

### 5.4.3 Multi-valued Consensus

The third option for implementing infrastructure-assisted consensus is at the level of *multi-valued* consensus. This section presents such a hybrid multi-valued consensus algorithm that makes use of an infrastructure service if it is available. In addition to the properties defined in the system model, the algorithm provides the following two properties:

- **Fast Termination 1:** If all correct nodes have the same initial value, the algorithm terminates within a constant, small number of steps.
- **Fast Termination 2:** If the infrastructure is available, all correct nodes decide within a constant, small number of steps.

Figure 14 shows a pseudo-code definition of our consensus algorithm. The algorithm is inspired by the multi-valued consensus algorithm defined by Correia et al. [12]. The two main differences to the original algorithm are the interaction step with the infrastructure, and a reduction of message size. Due to space limitations, we do not present a rigorous correctness proof here, but only give an informal discussion of the algorithm.

At the beginning, all nodes broadcast their local value to all nodes including the central server  $p_s$ . Next,  $p_s$  selects a value (which matches the value of the correct nodes if all of them propose the same value) and broadcasts this value with a *justification* (lines 14/15). The *justification* of a value  $v$  is the set of  $f+1$  nodes from which an INIT message for  $v$  has been received. The *justification* of  $\perp$  is a set of  $n-f$  nodes, in which no  $f+1$  nodes propose the same value. The justification set of a value can efficiently be encoded by  $n$  bits (1 bit per node). This compact justification is a significant difference to the original algorithm [12], which instead used a vector of  $n-f$  received messages for justifying values. The bit encoding is sufficient to validate messages, and is more appropriate for embedded systems, as it significantly reduces the amount of network traffic and the demand for local memory.

If all correct nodes are able to timely interact with a correct  $p_s$ , they all obtain an identical, justified COORD value. A COORD value is *justified* if it contains a justification set such that an INIT message has been received from all of its members, and the messages from the justification set prove the validity of the value chosen by the coordinator (i.e., for a value  $v \neq \perp$ , there is a set of  $f+1$  identical messages, and for  $v = \perp$ , there is a set of  $n-f$  message without a subset of  $f+1$  identical messages). If the interaction with  $p_s$  fails, a node calculates its own justified value.

Next, every node broadcasts a PRE message with a value and its justification obtained in the step before. In line (7) the justification of all PRE messages can be verified as described for the COORD value. Note that it may happen that a PRE message is not justified when it is delivered (as a required INIT message might not yet have been received), but it may later become justified through the reception of the missing INIT message. The justification ensures that if all correct nodes propose the same value  $v_1$ , then no other value  $v_2 \neq \perp$  may get a justification (as this would require the justification from  $f+1$  nodes), nor may  $v_2 = \perp$  get a justification (as any  $n-f$  nodes contain at least  $n-2f$  correct nodes (i.e., with  $n \geq 3f+1$ , at least  $f+1$  correct nodes).

Line (8) ensures that if a node selects  $b=1$  because of PRE messages for a value  $x_1$ , no other node selects  $b=1$  for a different value  $x_2$ . If one node selects  $x_1$ , there are at least  $f+1$  PRE messages for this value, so all other nodes receive at least one of these messages. If all propose the same value, there will be only PRE messages for that value, causing all correct nodes to propose  $b=1$ .

We assume the use of a randomized binary consensus such as that of Bracha [7], which guarantees that if all correct nodes start a round with identical values, they decide in the same round. The two fast termination properties follow directly from this property and the observation that if either all correct nodes have identical initial values or all of them successfully interact with the infrastructure, they all propose the same value ( $b=1$ ) to binary consensus.

The selection of the TIMEOUT value (in line 2) has a direct impact on the efficiency of the algorithm. A TIMEOUT value too short will cause the failure of infrastructure interaction. In this case, the algorithm will work only with the weaker guarantees of distributed consensus without infrastructure. A large TIMEOUT value will delay consensus execution for a large period of time in case that the infrastructure is unavailable or corrupt.

#### 5.4.4 Final remarks

Comparing the three approaches, the integration of infrastructure interaction at the multi-valued consensus level seems to be the most promising. If the infrastructure is available in this step, all correct nodes will propose the same value to binary consensus, ensuring fast termination. This variant could be combined with infrastructure-assisted binary consensus. The combination ensures fast termination in case that the infrastructure is unavailable at the start of the consensus, but becomes available later during the binary consensus phase. In addition, the infrastructure interaction at the reliable multicast level would help to reduce the total number of message, but only if the infrastructure does not show malicious behaviour.

## 5.5 Summary

In this section, we have discussed approaches for efficiently solving the consensus problem in distributed embedded systems in realistic environments in which an ad-hoc network and an infrastructure network are simultaneously available. The general idea is that if the infrastructure is not available, the participants execute a randomized consensus algorithm with probabilistic termination guarantees using the ad-hoc network. If the infrastructure is available, the participants take advantage of this and achieve consensus with better termination guarantees.

This section has presented on-going work. The presented ideas still lack an experimental validation, which should provide real data about the relative behaviour of the proposed approaches and of previously published consensus algorithms. This validation is currently being done by means of simulation, and it is expected that some results will be available still in the course of the HIDENETS project. These simulation results may be provided in the scope of future deliverables, namely deliverable D4.2.2 (“Application of the evaluation framework to the complete scenario”).

Regarding the implementation and integration of this service within the HIDENETS architecture, we believe that this would not be a difficult task, since the Intrusion Tolerant Agreement service, and in particular the infrastructure-assisted improvement proposed here, do not depend on other services. As shown in the previous sections, the solution is fundamentally based on improved algorithmic solutions, and is just dependent on the existence of multiple network connections (at least a connection within the ad-hoc domain, and possibly an additional connection to the infrastructure domain).

An issue to be investigated in the future is using distributed access points instead of a central server to support the consensus progress. Furthermore, the memory consumption of the protocol, for example for communication buffers, is an important issue in embedded systems. This aspect of the protocol should be accurately examined, and the worst-case memory consumption of the protocols should be minimised. We strongly believe that tailored consensus solution will help to construct dependable distributed embedded systems.

## 6 Concluding remarks

This deliverable describes research work developed in HIDENETS, concerned with improving middleware resilience services while assuming operation not only in the ad-hoc domain, but also in the infrastructure domain. This deliverable should be understood as a complementary deliverable to D2.3 (Service level resilience solutions for the ad-hoc domain), where we addressed solutions for the considered HIDENETS services considering the operation in the ad-hoc domain only. In HIDENETS we consider different realms of operation: operation in the ad-hoc and operation in the infrastructure domain. We also consider the possibility of an integrated operation in both domains.

It is important to note that the contributions provided in this deliverable have been developed without a specific intention of including them in any of the proof-of-concept prototypes being developed in the scope of WP6. Because of that, the description that we provide in this deliverable is not focused on the implementation, but rather on the detailed description of the proposed solutions, and in some cases, on their evaluation.

There are three contributions provided in this deliverable. The first one concerns a Fault Detection and Diagnosis (FDD) middleware component that may be used to manage faults that may cause an end-to-end service failure. The basic idea is to use Bayesian Networks for fault diagnosis, with the models being parameterized using observations from the network. Another contribution consists of a study of a solution for optimizing the dependability/performance trade-off for replicated servers in the infrastructure domain. The solution could be implemented as a sub-service of the Reconfiguration Manager, which is discussed in this deliverable, in Section 4.4. Finally, the deliverable also provides a study on the possibility of improving the Intrusion Tolerant Agreement service, by considering a solution that exploits the availability of servers in the infrastructure domain.

We believe that these contributions are an interesting complement of the work concerning the development of resilient solutions for the ad-hoc domain that is presented in D2.3 [11]. Some specific open issues have been discussed in each section. For instance, in the case of the proposed improvements for the Intrusion Tolerant Agreement service, several possibilities have been mentioned, but only one has been addressed more carefully. On the other hand, in this case evaluation work is still needed and, more generally, full implementations and evaluation through proof-of-concept setups can also be considered an open point.

## References

- [1] M. K. Aguilera and S. Toueg. Failure detection and randomization: A hybrid approach to solve consensus. *SIAM J. Comput.*, 28(3):890–903, 1999.
- [2] D. Angluin, M. J. Fischer, and H. Jiang. Stabilizing consensus in mobile networks. In *DCOSS*, pages 37–50, 2006.
- [3] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING*, 1(1):11–33, 2004.
- [4] N. Badache, M. Hurfin and R. Macêdo. Solving the consensus problem in a mobile environment. In *IPCCC*, pages 29-35, IEEE, 1999.
- [5] D. Barman and I. Matta. A bayesian approach for tcp to distinguish congestion from wireless losses. Technical report, Department of Computer Science - Boston University, 2003.
- [6] M. Ben-Or. Another advantage of free choice (extended abstract): Completely asynchronous agreement protocols. In *PODC '83*, pages 27–30. ACM Press, 1983.
- [7] G. Bracha. An asynchronous  $[(n - 1)/3]$ -resilient consensus protocol. In *Proceedings of the third annual ACM symposium on Principles of distributed computing*, pages 154–162. ACM Press, 1984.
- [8] I. de Bruin et al, Specification HIDENETS laboratory set-up scenario and components, EU FP6 IST project HIDENETS, deliverable D6.2, October 2007.
- [9] A. Casimiro et al, Resilient Architecture (Final Version), EU FP6 IST project HIDENETS, deliverable D2.1.2, December 2007.
- [10] C. Cachin, K. Kursawe, and V. Shoup. Random oracles in Constantinople: Practical asynchronous Byzantine agreement using cryptography. *Journal of Cryptology*, 18(3):219–246, July 2005.
- [11] A. Casimiro et al, Service level resilience solutions for the ad-hoc domain, EU FP6 IST project HIDENETS, deliverable D2.3, June 2008.
- [12] M. Correia, N. F. Neves, and P. Veríssimo. From consensus to atomic broadcast: Time-free byzantine resistant protocols without signatures. *Computer Journal*, 41(1):82–96, Jan 2006.
- [13] M. Correia, P. Veríssimo, N. F. Neves. The design of a COTS real-time distributed security kernel. In *Fourth European Dependable Computing Conference*, Toulouse, France, October 2002.
- [14] V. Drabkin, R. Friedman, and M. Segal. Efficient byzantine broadcast in wireless ad-hoc networks. In *Proc. of the 2005 Int. Conf. on Dependable Systems and Networks*, 160–169, 2005.
- [15] L. Falai et al, Mechanisms to provide strict dependability and real-time requirements, EU FP6 IST project HIDENETS, deliverable D3.3, June 2008.
- [16] M.J. Fischer, N.A. Lynch, M.S. Paterson. Impossibility of Distributed Consensus with One Faulty Process. *JACM*, 32(2):374–382, 1985.
- [17] D. Heckerman. A tutorial on learning with Bayesian networks. Technical report, Microsoft Research, Advanced Technology Division, March 1995.
- [18] C. Hood and C. Ji. Probabilistic network fault detection. *Global Telecommunications Conference*, 3, 1996.
- [19] F. V. Jensen. *Bayesian Networks and Decision Graphs*. Springer-Verlag New York, Inc., 2001.
- [20] J. Liu, I. Matta, and M. Crovella. End-to-End Inference of Loss Nature in a Hybrid Wired/Wireless Environment. *Proceedings of WiOpt03: Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks*, 2003.
- [21] P. Lollini et al. Application of the evaluation framework to the complete scenario (preliminary version). EU FP6 IST Project HIDENETS, deliverable D4.2.1, Dec. 2007.

- [22] S. McCanne, S. Floyd, K. Fall, K. Varadhan et al. *Network simulator - ns2*, <http://www-mash.cs.berkeley.edu/ns/>, 1997.
- [23] A. Mostefaoui, M. Raynal, and F. Tronel. The best of both worlds: A hybrid approach to solve consensus. In *Proc. of the 2000 Int. Conf. on Dependable Systems and Networks*, pages 513–522, 2000.
- [24] K. P. Murphy. *Dynamic Bayesian Networks: Representation, Inference and Learning*. PhD thesis, University of California, Berkeley, 2002.
- [25] A. Nickelsen and J. Grønbæk. Probabilistic fault detection in network communication. Technical report, Aalborg University, May 2006.
- [26] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, California, 1988.
- [27] K. Przytula and D. Thompson. Construction of Bayesian networks for diagnostics. *Aerospace Conference Proceedings, 2000 IEEE*, 5, 2000.
- [28] M. Radimirsch and et al. *D1.1: Use case scenarios and preliminary reference model*. Highly Dependable ip-based NETWORKS and Services – HIDENETS, <http://www.hidenets.aau.dk/>, 2006.
- [29] J. Rosenberg et al., “SIP: Session Initiation Protocol,” IETF, RFC3261, June 2002.
- [30] H. Seba, N. Badache, and A. Bouabdallah. Solving the consensus problem in a dynamic group: An approach suitable for a mobile environment. In *Proc. of the 7th Int. Symp. on Computers and Communications*, page 327. IEEE Computer Society, 2002.
- [31] Sophia-Antipolis. Ns simulator for beginners. Technical report, Univ. de Los Andes, Mérida, Venezuela and ESSI, 2003.
- [32] M. Steinder and A. Sethi. The present and future of event correlation: A need for end-to-end service fault localization. *World Multi-Conf. Systemics, Cybernetics, and Informatics*, pages 124–129, 2001.
- [33] M. Steinder and A. Sethi. Probabilistic fault localization in communication systems using belief networks. *IEEE/ACM Transactions on Networking (TON)*, 12(5):809–822, 2004.
- [34] R. Stewart, et al., “Aggregate Server Access Protocol (ASAP),” IETF, draft-ietf-rserpool-asap-11.txt, June 2006.
- [35] R. Stewart, et al., “Endpoint Handlespace Redundancy Protocol (ENRP),” IETF, draft-ietf-rserpool-enrp-11.txt, June 2006.
- [36] M. Suojanen, S. Andreassen, and K. Olesen. A method for diagnosing multiple diseases in MUNIN. *Biomedical Engineering, IEEE Transactions on*, 48(5):522–532, 2001.
- [37] M. Tuexen, et al., “Requirements for Reliable Server Pooling,” IETF, RFC3237, January 2002.
- [38] P. Verissimo and L. Rodrigues. *Distributed systems for system architects*. Kluwer Academic Publishers, 2001.
- [39] P. Verissimo. Travelling through wormholes: Meeting the grand challenge of distributed systems. In *Proc. Int. Workshop on Future Directions in Distributed Computing*, pages 144–151, Bertinoro, Italy, June 2002.
- [40] P. Verissimo, A. Casimiro. The Timely Computing Base model and architecture. *IEEE Transactions on Computers*, 51(8):916–930, 2002.
- [41] S. C. Wang, W. P. Yang, and C. F. Cheng. Byzantine agreement on mobile ad-hoc network. 2004 *IEEE Int. Conf. on Networking, Sensing and Control*, 1:52–57, Mar. 2004.
- [42] G. Weiss and F. Provost. The effect of class distribution on classifier learning. Technical report, Dept. Comput. Sci., Rutgers Univ., New Brunswick, NJ, 2001.
- [43] W. Wu, J. Cao, J. Yang, and M. Raynal. A hierarchical consensus protocol for mobile ad hoc networks. In *PDP '06: Proc. of the 14th Euromicro Int. Conf. on Parallel, Distributed, and Network-Based Processing*, pages 64–72. IEEE Computer Society, 2006.