UNIVERSITÀ DEGLI STUDI DI FIRENZE
Dipartimento di Sistemi e Informatica
Dottorato di Ricerca in Informatica e Applicazioni
XXII Ciclo
Settore Disciplinare INF/01

CRITICAL INFRASTRUCTURES: A CONCEPTUAL FRAMEWORK FOR DIAGNOSIS, SOME APPLICATIONS AND THEIR QUANTITATIVE ANALYSIS

ALESSANDRO DAIDONE

Supervisor: *prof. Andrea Bondavalli*

PhD Coordinator: *prof. Rocco De Nicola*

December, 2009

To Tiziana
. . . and Arianna/Gabriele

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Critical infrastructures are becoming more and more open and pervasive, increasing their complexity and incorporating previously disjoint systems to provide critical services; these infrastructures are often composed of unstable COTS components or old legacy machinery, which were designed without considering their possible growth, the possible use of new technologies and their interconnectedness. All this elements expose critical infrastructures to risk of mismanagement errors caused both by accidental faults and by malicious attacks and intrusions, so it is necessary to find methods and solutions for assuring their resiliency.

Traditional diagnostic solutions are based on static assumptions about system behavior, fault model and detection mechanism (the "unusual" component behavior can be defined a-priori and assumed due to faults in the component itself). Critical infrastructures comprise more dynamic scenarios, where failure definitions and/or system specifications change over time, so that traditional diagnostic solutions cannot be applied as they are.

This work proposes a conceptual framework for on–line system diagnosis, taking into account both the local and the global point of view: each system node diagnoses both itself (local diagnosis) and remote nodes (private diagnosis) based on the local perception of their behavior; when some relevant events happen, distributed diagnosis is run to reach consensus about the diagnosis of a remote node. Diagnosis is based on the observation of system behavior over time (monitoring), collecting information at different architectural levels and correlating together diagnostic judgments inferred from lower architectural levels and relevant events observed somewhere in the system. The proposed framework is then instanced within the design of two different architectures: the CRUTIAL architecture, devoted to the protection of a SCADA–based critical infrastructure, and the HIDENETS architecture, devoted to the provision of distributed services for adaptive operation.

Some evaluation work is then presented, aiming to analyze some of the diagnosis and reconfiguration strategies adopted in the selected architectures. A quantitative analysis of the redundant architecture of the nodes protecting the CRUTIAL system is presented; those nodes are rejuvenated by proactive and reactive recoveries: the objective of the analysis is to identify the relevant parameters of the architecture, to evaluate how effective is the trade-off between proactive and reactive recoveries, and to find the best parameter setup. Another quantitative analysis is presented, aiming to sustain a fundamental assumption of the recovery operations: not-increasing probability of intrusion over time. Then a quantitative analysis of a few failure detection and reconfigura-

tion strategies for the management of a replicated server pool is presented; the objective is to understand the impact of the parameters of the strategies and to obtain the optimal parameter configuration for a selected set of fault scenarios.

Finally an ongoing work for enhancing the security in SCADA infrastructures is presented; this work is performed in the context of the INSPIRE project and focuses on security at communication layer. An on–line diagnostic mechanism is sketched for the detection of sinkhole attacks to the routing protocol of a wireless sensor network in a SCADA island.

ACKNOWLEDGMENTS

The first person I would like to thank is my tutor prof. Andrea Bondavalli; it was a great experience working with him, both from the professional and personal point of view.

Then I would like to thank all the other people that helped me, in may ways, to conclude my PhD experience: all the members of the RCL research group, people at the DSI department of the University of Florence (especially the PhD students), people from the DCL group at the ISTI institute of CNR (especially Felicita di Giandomenico), some of the visiting students the RCL group invited in the last years (especially Thibault Renier).

I would also like to thank the reviewers of this thesis, prof. Luigi Romano (Università degli Studi di Napoli Parthenope) and prof. Hans Peter Schwefel (Aalborg University), for their precious comments and suggestions.

Last, but not least, an enormous thank goes to my wife Tiziana, with whom I'm dividing all the greatest experiences of my life.

This work has been partially supported by the European Community through the following research actions: the FP6 IST project CRUTIAL (contract n. 27513), the FP6 IST project HIDENETS (contract n. 26979), the FP6 IST network of excellence RESIST (contract n. 26764) through the FOREVER project, and the FP7 ICT-SEC Project INSPIRE (contract n. 225553).

INTRODUCTION

What CIs are

Modern society has reached a point where everyday life heavily relies on the reliable operation and intelligent management of infrastructures such as electric power systems, telecommunication networks, water distribution networks, transportation systems, etc. Those infrastructures are actually *critical* infrastructures (CIs): people expects that the services provided by those infrastructures are always available (24/7) and relies on those services.

Critical infrastructures are large–scale, complex, interconnected and distributed systems. The interconnectedness of critical infrastructures makes them vulnerable to failures, which may be caused by natural disasters, accidental failures, human errors or malicious attacks; the consequences of those failures could be tremendous on several perspectives: societal consequences, health hazards, economic effects. On the other hand, their interconnectedness also allows a level of operational redundancy for fault tolerance, provided it can be exploited effectively!

Just to give an idea of the consequences of failures in critical infrastructures, two incidents involving the electric power grid are reported hereafter.

The European blackout in 2006

On 4th November 2006 a local fault in Germany's power grid cascaded through several countries in Europe, resulting in parts of six countries left without electric power for a couple of hours: Germany, France, Austria, Italy, Belgium, and Spain (about 10 million people). The fault cascaded through several countries because the European electricity grid is an interconnected network encompassing the European countries, with each sub network being managed and controlled by the respective national TSO[1].

The fault originated from northern Germany, where a high–voltage line had to be switched off by the local TSO to let a ship pass underneath the line through the Ems river; this event lead to the overloading of the neighbor lines and finally to the splitting of the UCTE[2] interconnected network into three zones: west, east and south–east. The western zone lacked power, while the eastern zones had too much power: to cope with this lack of power the automatic devices in the western zone had to switch the power off in the above mentioned countries.

After the incident the UCTE was delegated to produce a report about the incident; the report [UCTE 06] identified two main causes for the incident:

1 Transmission System Operator
2 Union for the Coordination of Transmission of Electricity

1. Non fulfillment of the $n-1$ criterion[3]: the German grid was not fulfilling the $n-1$ criterion after the manual disconnection of the high–voltage line.

2. Insufficient inter–TSO coordination: the initial planning for switching–off the high–voltage line was correctly prepared among the involved TSOs, but the last minute change in the time for the maneuver was communicated too late from the German operator.

The European blackout in 2006 was hence caused by a bad management of the infrastructure, especially taking into account that the beginning of the cascading failure was provoked by a planned event, not an unexpected one!

Another incident involving the electric power grid which had even more serious consequences than the above mentioned one was the blackout in the US in 2003.

The north–east US blackout in 2003

On August 14th 2003 an electric power blackout occurred in large portions of the mid–west and north–east United States and Ontario (Canada), affecting about 50 million people; the power grid was restored after a week, causing tremendous social and economic consequences (e.g. failure of traffic lights, people trapped in lifts and underground railways, dead refrigerators, social panic).

Immediately after the blackout, the US–Canada Power System Outage Task Force was established to investigate the causes of the blackout and to recommend actions that would minimize the possibility of similar, future events; the result was a report [Task Force 04] which identified that the incident was caused by the concurrent occurrence of the following factors:

1. Increment in energy demand due to the use of air conditioners (the temperatures were above the average in that period); the increment in energy demand was considered manageable;

2. Unexpected loss of some generating and transmission capacity;

3. Inadequate tree trimming;

4. Poor information awareness due, in part, to computer and network malfunctions at the operator controlling the grid in northern Ohio.

The report underlined that the malfunction of some critical equipment (possibly as a result of inadequate diagnostic support) and the behavior of protective devices complicated the management of the events; moreover, the analysis suggested that the blackout was not only due to the negligence of one company, but also to more pervasive structural and technical problems.

How CIs were developed

Most of the large–scale infrastructures have been developed connecting previously stand–alone systems, usually developed from proprietary architectures

3 The $n-1$ criterion deals with the ability of the transmission system to lose a linkage without causing an overload failure elsewhere.

where ad hoc solutions were chosen and several (electronic) components were developed independently [Simoncini 04, Egan 07].

Components of the infrastructure were usually ad–hoc components which were designed having in mind the entire infrastructure; this approach has the following positive and negative aspects.

Positive aspects are related to the fact that the system designer has the complete knowledge about the system and there are no third–party components which could make harder the procurement and the validation of parts of the system (which is mandatory for a safety–critical system); system re–design and updates do not depend on third–parties and are easier to validate.

Negative aspects derive from the fact that several components may be soon obsolete, given the rapid evolution of technologies, and there could be the need for upgrading some of them, specially if the operational life of the system is rather extended. Another disadvantage derives from the strict dependence between components and the system itself (through the design), which makes the system rather inflexible and not adaptable to different contexts or to be interfaced to other systems. Moreover, when an upgrade or a revision is needed, a new validation activity needs to be performed.

How CIs are changing

Due to technological advances, deregulations and market liberalizations, critical infrastructures are progressively becoming larger in scale and more complex, making their management more challenging: critical infrastructures were not designed to become so large, they evolved due to growing demand, but the way in which they are now operated is often beyond the original design parameters [Johnson 07].

Moreover it's harder than before to understand how these infrastructures work and which are the interactions between all the components [Egan 07]. Critical infrastructures are also becoming more heterogeneous and distributed, given that they are growing often integrating previously disjoint systems, which were not designed to be interconnected: this makes it more difficult to secure reliable operation.

As technology advances, the interactions in the reliable operation and management between different infrastructures are becoming increasingly more important. For example, there are strong links between the reliable operation of the electrical power grid and the communication networks, as well as between the electrical power grid and the water networks.

Critical infrastructures have many common characteristics and requirements, they share the underlying dynamics and suffer similar impacts from failures. All this elements naturally point to the need for a common methodological framework for modeling their behavior and for designing intelligent monitoring and diagnostic mechanisms able to detect and diagnose both accidental and malicious faults, triggering the proper recovery actions.

How CIs should be designed

Critical infrastructures should be designed/updated so that they could have the following characteristics [Simoncini 04]:

- Shift (part of) the reliability and safety properties from the components toward the architectural design: techniques for monitoring, diagnosis and recovery should be as most as possible independent from the specific components.
- Use of generic components (possibly COTS[4]): generic components can be easily substituted (e.g. to follow technological evolution) without re–design or re–validation of the entire infrastructure.
- Use of a hierarchical approach for functional and non functional properties: this approach tends to ease the validation of the system.
- Make the system adaptive to unforeseen changes that can occur at run–time: the system should be able to adapt to natural system's evolution, to the occurrence of fault patterns different from those foreseen at design time, or the change of application's dependability requirements during the operational lifetime.

Recent research activity in CIs

The interest in research about critical infrastructures is currently very active worldwide; focusing just on research funded by the European commission on ICT[5] thematics, there are several examples to cite. The European community has recently funded the FP6[6] and is currently funding the FP7[7] framework programmes for research and technological development in several areas, among which our interest is in the ICT, IST[8] and SEC[9] thematic priorities.

Recent examples of research projects dealing with SCADA[10] infrastructures related to the electrical power grid are the following: CRUTIAL[11] (IST–FP6, ended in 2008), which addressed new networked ICT systems for the management of the electric power grid, and IRRIIS[12] (IST–FP6, ended in 2009), which worked for increasing the availability, survivability and resilience of critical power and communication infrastructures.

Other examples of research actions related to power infrastructures are GRID[13] (IST–FP6, ended in 2008), which worked for increasing the understanding of the vulnerability of the integrated power system and the controlling ICT systems, and IntelliCIS[14] (COST[15], from May 2009 to May 2013), which will develop innovative intelligent monitoring, control and safety methodologies for critical

4 Commercial Off–The–Shelf
5 Information and Communications Technology
6 Sixth Framework Programme
7 Seventh Framework Programme
8 Information Society Technologies
9 Security
10 Supervisory Control And Data Acquisition
11 CRitical UTility InfrastructurAL Resilience
12 Integrated Risk Reduction of Information–based Infrastructure Systems
13 GRID is a coordination action funded by the IST-FP6
14 Intelligent Monitoring, Control and Security of Critical Infrastructure Systems
15 European Co-Operation in Science and Technology

infrastructure systems, such as electric power systems, telecommunication networks, and water distribution systems.

Examples of research activity not directly related to electric power infrastructures, but still related to critical infrastructures, are INSPIRE[16] (ICT–SEC–FP7, ongoing), which is addressing the problem of increasing the security and the protection of SCADA–based infrastructures at communication layer, and HIDENETS[17] (IST–FP6, ended in 2008), which developed resilient architectural solutions for adaptive operation in automotive communications (both car–to–car and car–to–infrastructure scenarios).

In this context, we focus our analysis on i) monitoring and diagnosis and ii) analysis and evaluation of the resiliency and dependability achieved.

CI characteristics affecting diagnosis

Diagnostic solutions should take into account that critical infrastructures, as they are evolving, are:

- Large, open and pervasive, so individual parts of the infrastructure could not even be aware of the entire infrastructure;
- Highly dynamic, time–varying and uncertain; wireless context exacerbate these characteristics (e.g. some wireless entities can suddenly join or leave the infrastructure);
- Non homogeneous, so different parts of the infrastructure may be very diverse one from the other for several reasons: e.g. because of hardware, software, performance levels, resiliency levels, services provided, connections or synchronization mechanisms;
- Made up of COTS and legacy sub–systems, so the system designer has limited knowledge and control over them (goodness of these large grained components could be related to the quality of the service provided, rather than to the absence of faults).
- Data–rich environments, so methods are necessary to collect and filter information relevant for diagnosis purposes.

All this makes diagnosis a very difficult activity, because system observation and monitoring need to be performed through detection devices with unknown characteristics related to coverage, false positive probabilities etc. Moreover it is not always possible to introduce new detection/monitoring devices within the system to feed diagnosis, e.g. when old legacy sub-system are integrated in the infrastructure.

Not only faults. . .

In a wider perspective, diagnosis needs to assess the suitability of component-/sub–system/infrastructure to provide services with adequate quality, which may dynamically change over time. In this view, the goodness of a component

16 INcreasing Security and Protection through Infrastructure REsilience
17 HIghly DEpendable ip–basedNETworks and Services

is not strictly tied to the absence or presence of faults which may impair its functionality; rather, it is the overall quality of service which determines whether a component is useful and contributes to the system activities or it is better to keep it out. Actually, this more general framework allows to capture several possible scenarios, such as:

1. The component QoS[18] decreases because of malfunctions affecting the component itself;
2. The application using the component changes its QoS requirements in such a way that they do not match anymore with the specification of the component under utilization;
3. Changes in the environment (e.g., system load) may lead to a change in the QoS provided.

Component obsolescence is a typical example of case 2, while classical examples of case 1 can be taken from the system fault tolerance area.

18 Quality of Service

1 STATE OF THE ART AND OPEN PROBLEMS

This chapter introduces the concept of diagnosis in dependable systems, describing the complete chain starting from the observation of the behavior of the monitored component to the definition of a diagnostic judgment.

An overview of the basic concepts of dependability is first presented, so that all the terms related to the diagnosis problem are defined. Then the definition of diagnosis is given, and the properties of a diagnostic mechanism are presented. A general framework to model over–time diagnosis scenarios is then presented; this framework takes into account all the involved system aspects (the monitored component, the deviation detection mechanism and the state-diagnosis mechanism) accounting for them separately.

An overview of the traditional diagnostic mechanisms is presented, discussing the differences between off–line and on–line mechanisms, and between heuristic and probabilistic approaches. The most promising on–line mechanisms are described: α–count among the heuristic approaches and the diagnosis based on hidden Markov models among the probabilistic approaches. A short discussion about advantages and disadvantages between the two approaches is presented.

The last part of the chapter introduces the need for new diagnostic tools able to run in more dynamic scenarios, where the amount of indicators to observe is very large and information correlation becomes a key activity. An overview of some monitoring tools is presented, and the rationale behind a diagnostic mechanism based on event correlation is described.

1.1 BASIC CONCEPTS OF DEPENDABILITY

This section presents a basic set of definitions that will be used throughout the entire thesis; these definitions are related to the basic concepts of dependability and security applied to computer-based systems.

When dealing with dependability and security of computing and communication systems, the reference taxonomy and definitions are those given in [Avižienis 04]: this work is the result of a work originated in 1980, when a joint committee on "Fundamental Concepts and Terminology" was formed by the TC on Fault–Tolerant Computing of the IEEE CS[1] and the IFIP WG 10.4

1 This is the technical committee (TC) on Fault–Tolerant Computing of the Institute of Electrical and Electronic Engineers (IEEE) Computer Society; IEEE is an international non-profit, professional organization for the advancement of technology related to electricity.

"Dependable Computing and Fault Tolerance"[2] with the intent of merging the distinct but convergent paths of the dependability and security communities.

A system and its functionalities

Let's start our overview about the basic concepts of dependability by defining what a computer-based systems is. [Avižienis 04] defines a *system* as an entity that interacts with other entities, i.e., other systems, including hardware, software, humans, and the physical world with its natural phenomena. From a structural viewpoint, a system is composed of a set of components bound together in order to interact, where each component is another system, etc. The recursion stops when a component is considered to be atomic. The *function* of such a system is what the system is intended to do[3], whilst the *behavior* of a system is what the system does to implement its function[4]. The *service delivered* by system S is its behavior as it is perceived by its user(s); a user is another system that receives service from system S. *Correct service* is delivered when the service implements the system function.

Threats to dependability

A service failure, abbreviated here to *failure*, is an event that occurs when the delivered service deviates from correct service. The deviation from correct service may assume different forms, which are called service *failure modes* and which are ranked according to failure severities (e.g. minor vs. catastrophic failures). Since a service is a sequence of system's external states, a service failure means that at least one (or more) external state of the system deviates from the correct service state: the deviation is called *error*. The adjudged or hypothesized cause of an error is called *fault*[5]; a fault is *active* when it causes an error, otherwise it is *dormant*. When the functional specification of a system includes a set of several functions, the failure of one or more of the services implementing the functions may leave the system in a *degraded mode*, which still offers a subset of needed services to the user.

Chain of threats

The creation and manifestation mechanisms of faults, errors, and failures, depicted in figure 1.1, is called "chain of threats". The chain of threats summarizes the causality relationship between faults, errors and failures. A fault activates in component A[6] and generates an error; this error is successively transformed into other errors (*error propagation*) within the component (internal propagation) because of the computation process. When some error reaches the service interface of component A, it generates a failure, so that the service delivered by A to component B becomes incorrect. The ensuing service failure of component A appears as an external fault to component B.

2 The International Federation for Information Processing (IFIP) is an umbrella organization for national societies working in the field of information technology; the 10.4 working group (WG) is part of the technical committee about "Computer Systems Technology".

3 The function of a system is described by the functional specification in terms of functionality and performance.

4 The system behavior is described by a sequence of states.

5 Faults can be internal or external to the system.

6 The fault could be 1) an internal fault that was previously dormant and that has been activated by the computation process or environmental conditions, or 2) an external fault.

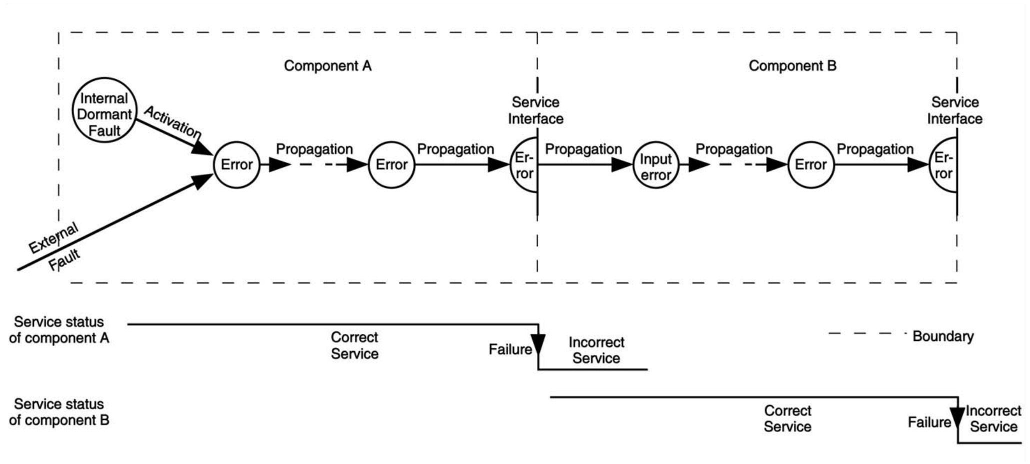


Figure 1.1: The chain of threats: a fault in component *A* activates and generates an error; errors propagate until component *A* fails; the failure of *A* appears as an external fault to *B*.

What is dependability

After having defined what is a system, its correct service and the threats which can affect the service, it is time to give the definition of dependability. The original definition of *dependability* is the following: "dependability is the ability to deliver service that can justifiably be trusted"; given that this definition stresses the need for justification of "trust", an alternate definition is given in [Avižienis 04]: "dependability of a system is the ability to avoid service failures that are more frequent and more severe than is acceptable". This last definition has a twofold role, because in addition to the definition itself it also provides the criterion for deciding whether the service is dependable or not.

Dependability attributes

Dependability is an integrating concept which encompasses the following attributes:

AVAILABILITY: readiness for correct service.

RELIABILITY: continuity of correct service.

MAINTAINABILITY: ability to undergo modifications and repairs.

SAFETY: absence of catastrophic consequences on the user(s) and on the environment.

INTEGRITY: absence of improper system state alterations.

When addressing security, an additional attribute need to be considered: *confidentiality*, which is the absence of unauthorized disclosure of information.

Based on the above definitions, *security* is defined as the composition of the following attributes: confidentiality, integrity, and availability; security requires in effect the concurrent existence of availability for authorized actions only, confidentiality, and integrity (with "improper" meaning "unauthorized").

Means to attain dependability

The means to attain dependability and security are grouped into four major categories:

FAULT PREVENTION: means to prevent the occurrence or introduction of faults.

FAULT TOLERANCE: means to avoid service failures in the presence of faults.

FAULT REMOVAL: means to reduce the number and severity of faults.

FAULT FORECASTING: means to estimate the present number, the future incidence, and the likely consequences of faults.

The focus here is mainly on fault tolerance.

Fault tolerance

[Avižienis 04] explains that fault tolerance is carried out via error detection and system recovery, that is by first identifying the presence of an error (*error detection*), and then transforming a system state containing one or more errors and (possibly) faults into a state without detected errors and without faults that can be activated again (*fault handling*).

Diagnosis

Fault handling involves in turn *fault diagnosis*, that is the identification of the cause(s) of error(s) in terms of both location and type. Fault diagnosis is the key activity to perform *isolation*[7] of the faulty component from further participation in service delivery and *reconfiguration*, which either switches in spare components or reassigns tasks among the not failed components.

Properties of diagnostic mechanisms

Diagnostic mechanisms can be evaluated analyzing some of their properties, which are presented hereafter. Some of the above properties are defined in terms of probabilistic formulas, using the following notations: let I be the event "the component is identified as faulty" and F be the event "the component is really faulty".

COMPLETENESS: the capability of identifying that the component is faulty, given that the component is really faulty; completeness is expressed by the following formula: $p(I|F)$.

ACCURACY: a diagnostic mechanism is accurate if the component identified as faulty is really faulty; accuracy is expressed by the following formula: $p(F|I)$. If the diagnostic mechanism is accurate, it does not identify a component as faulty when the component is not faulty.

PROMPTNESS: a measure on how quickly the diagnostic mechanism is able to identify a faulty component, given that the component is faulty.

ROBUSTNESS: the capability to correctly take into account the incompleteness of detection observation.

7 Isolation can be physical or logical.

An ideal diagnostic mechanism should both identify all the faulty components (completeness) and do not accuse healthy components (accuracy), doing it in the shortest time (promptness). The properties of the diagnostic mechanisms are used to choose at design time which diagnostic approach could be the best for a specific task.

1.1.1 *Diagnosis framework*

This section presents the general diagnostic framework proposed in [Daidone 06] for modeling over–time diagnostic scenarios. The diagnostic framework, de- *Diagnosis framework*

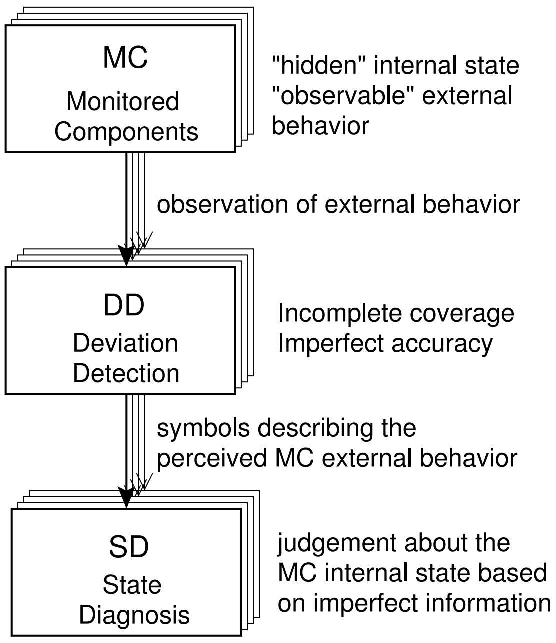

MC
Monitored Components
"hidden" internal state
"observable" external behavior
observation of external behavior
DD
Deviation Detection
Incomplete coverage
Imperfect accuracy
symbols describing the perceived MC external behavior
SD
State Diagnosis
judgement about the MC internal state based on imperfect information

Figure 1.2: The diagnosis framework identifying the chain constituted by the monitored component (MC), the error detection mechanism (DD) and the diagnostic mechanism (SD).

picted in figure 1.2, involves the following actors:

monitored component (mc). It is the system component under diagnosis. During system lifetime, the monitored component is affected by faults that might compromise its functional behavior.

deviation detection (dd). It is the entity which observes the external behavior of the monitored component and judges whether it is suitable or not. If the deviation detection mechanism has incomplete coverage and/or imperfect accuracy, it can raise *false positives* (when inexistent deviation is detected) and *false negatives* (when an existent deviation is not detected).

STATE DIAGNOSIS (SD). It is the entity that judges which is the internal state of the monitored component, based on information coming from the deviation detection mechanism. The state-diagnosis mechanism has to decide whether the services delivered by the monitored component continue to be beneficial or not for the rest of the system, deviations notwithstanding.

The external behavior of the monitored component is "observable", but the same observed unsuitable behavior could be caused by different faults, so the internal state of the monitored component[8] remains in some way "hidden"; moreover, unsuitable component behavior could also be determined by a change in the requirements of the user of the services delivered by the monitored component.

The diagnosis of the internal state of the monitored component is hence the result of the interpretation of deviation (error) detection information, which is in turn affected by incomplete coverage and imperfect accuracy. Whenever the diagnostic mechanism identifies that the monitored component deviates too much from its "appropriate behavior" (so that it could be no more useful for the system), specific alarms are raised in order to possibly trigger proper reconfiguration actions. For example, if the component is diagnosed as faulty, it is isolated and substituted by a spare backup; in this sense it is clear that false positives and false negatives are both undesired: false positives led to an early depletion of system resources, while false negatives drastically decrease the system dependability.

The diagnosis activity involves two information flows:

MC→DD: the deviation detection mechanism observes the external behavior of the monitored component;

DD→SD: the state-diagnosis mechanism collects deviation detections performed by the deviation detection mechanism

Each of the above information flows can be managed following either a proactive or a reactive schema: in the proactive schema, the entity that generates information sends it to the entity interested in it, whilst in the reactive schema the entity interested in fresh information explicitly asks for it.

[Romano 02] proposes three interaction patterns to manage the DD → SD information flow; the main differences between the alternative solutions are the granularity and frequency of the interactions:

CONTINUOUS MONITORING: there is an interaction every time the DD performs some detection.

8 The "internal state" is something related to the component situation with respect to faults; there is no relationship between this "internal state" and the possible component states related to the functionalities performed by the component itself.

buffered asynchronous monitoring: information about deviation detection is buffered by the DD and sent to the SD in chunks (the objective is to decrease interaction overhead).

failure triggered synchronous monitoring: the information about deviation detection is sent only when deviations are detected (so there is no interaction as long as no deviations are detected); after the detection of some deviation, the interaction becomes continuous for some time, until the SD judges that it is necessary.

The above solutions have different balances in terms of QoS vs. cost of feeding the data to the SD mechanism: "continuous monitoring" is the most costly and probably too aggressive in terms of the overhead it induces on the system; "buffered asynchronous monitoring" is cheaper than "continuous monitoring" for the interaction cost, but it requires storing capabilities in the DD mechanism and it negatively affects the promptness of the SD; "failure triggered synchronous monitoring" combines the advantages of "continuous monitoring" (timeliness of the input data and no need for storage) and of "buffered asynchronous monitoring" (reduction in communication cost), but can be impaired by omission faults in the DD.

1.2 traditional diagnosis

The first proposals for defining design guidelines for fault–tolerant systems date back to the late sixties: a significant example is [Avižienis 67], where the author introduced the concept of "fault–tolerant system", presented a classification of faults, and outlined techniques for masking, detection, diagnosis, and recovery. The author illustrated also the application of fault–tolerant criteria to the design of the experimental fault–tolerant JPL-STAR computer[9].

A large amount of literature has been produced since the late sixties on the classical problem of diagnosis in multiprocessors or embedded systems, where the target of diagnosis were sets of homogeneous, fine grained components; deep knowledge was usually available about the monitored components, allowing the assumption of simple fault models and failure semantics. Some relevant examples are described in the following.

[Preparata 67] is the first work which discussed the problem of system-level diagnosis, that is the problem of diagnosing systems composed of homogeneous units (e.g. a multi-processor system) connected by point–to–point, bidirectional links; the paper assumed the system was affected by multiple faults and proposed an automatic fault diagnostic mechanism based on a suitable set of tests between units (the collection of test results was called "syndrome" for

9 JPL-STAR is a spacecraft computer with long life and autonomy requirements and with strict weight and power constraints.

the system). [Preparata 67] introduced a simple graph-theoretic model for fault diagnosis, called "PMC model" (acronym derived from the surnames of the authors), which is still the subject of a lot of research work (e.g. [Manik 09]).

[Barsi 76] is another seminal work which is based on many premises of the PMC model; the "BGM model" (acronym again derived from the surnames of the authors) differs from the PMC model because of the interpretation of test results, simplifying the diagnostic procedure. Also the BGM model is still the subject of a lot of research work (e.g. [Vedeshenkov 02]).

In the above mentioned works diagnosis is based on the so called "one–shot diagnosis of a collected syndrome", so that the component is diagnosed as "failed" as soon as an error is observed, immediately triggering reconfiguration operations devoted to isolate, repair or replace the failed component.

Off–line and on–line approaches

For those cases in which one–shot diagnosis is not suitable several diagnostic methods have been proposed, which can be broadly classified into two groups: off–line analysis procedures, and on–line mechanisms. On–line approaches require that part of system resources (e.g. time) are dedicated to the diagnosing activity and, most important, require the use of not–destructive detection mechanisms; sometimes these requirements cannot be met. On the other hand, off–line approaches require that the monitored components are temporarily isolated from the system, this way depriving the system of some resources. [Coccoli 03] analyses the problem of choosing between an on–line or off–line approach for implementing the diagnostic mechanism in a control systems for critical applications.

The category of off–line analysis procedures includes more or less sophisticated procedures which analyze system error logs to derive trend analysis and predictions of future permanent faults, as proposed in [Lin 90, Iyer 90].

Heuristic and probabilistic approaches

On–line mechanisms work while the monitored system is working, so the diagnostic judgments can be used at run–time to act on the system configuration, aiming to tolerate the diagnosed faults. On–line diagnostic mechanisms include a variety of threshold–based over–time diagnostic mechanisms [Spainhower 92, Mongardi 93, Bondavalli 00, Bondavalli 04a, Daidone 06, Serafini 07, Nickelsen 09], using either heuristic or probabilistic approaches [Spainhower 92, Mongardi 93, Bondavalli 00, Bondavalli 04a, Daidone 06, Serafini 07, Nickelsen 09].

Heuristic approaches are typically simple mechanisms suggested by intuitive reasoning and then validated by experiments or modeling; for example, they count errors, and when the count crosses a pre–set threshold a permanent fault is diagnosed. An example of heuristic mechanism for the discrimination between transient and permanent faults is the α–count mechanism [Bondavalli 00] presented in section 1.2.1.

Probabilistic approaches are tailored to evaluate the probabilities of the monitored component being faulty, based on monitoring information collected. An example of diagnostic mechanism which follows the probabilistic approach is the mechanism based on HMM[10] [Daidone 06] presented in Section 1.2.2.

How to choose the more appropriate approach

A good reason to use a diagnostic mechanism based on a probabilistic approach is that a probabilistic approach uses the available knowledge about the system in the best way in order to improve diagnosis accuracy [Pizza 98]. It is worth noting that a trade–off exists between diagnosis accuracy and diagnosis promptness so that, in order to get better the first, the second could get worse and vice versa; moreover, heuristic approaches tend to be computationally lighter than the probabilistic ones.

1.2.1 *An heuristic approach for on–line diagnosis*

Heuristic approaches to diagnosis are typically based on simple mechanisms suggested by intuitive reasoning and then validated by experiments; for example, the so called "count–and–threshold" mechanisms count (in some way) error messages collected over time, raising alarms when the counter passes a given threshold. These approaches emerged in those application fields where components oscillate between faulty and correct behavior because a large fraction of faults are transient in nature; in those application fields one–shot diagnosis is not effective, because the cost for treating a transient fault as a permanent one is very disadvantageous.

The idea of thresholding the number of errors accumulated in a time window has long been exploited in IBM mainframes: in the earlier examples, as in IBM 3081 [Tendolkar 82], the automatic diagnostics mechanism uses the above criteria to trigger the intervention of a maintenance crew; in later examples, as in ES/9000 Model 900 [Spainhower 92], a retry–and–threshold scheme is adopted to deal with transient errors.

In the TMR[11] MODIAC railway interlocking system by Ansaldo, the architecture proposed in [Mongardi 93], two failures experienced in two consecutive operating cycles by the same hardware component, which is part of a redundant structure, make the other redundant components consider it as definitively faulty.

In [Sosnowski 94] big attention is given to the error detection and error–handling strategies to tolerate transient faults: a fault is labeled as transient if it occurs less than k times in a given time window.

The most sophisticate representative of the heuristic mechanisms is α–count [Bondavalli 00], an heuristic proposed for the discrimination between transient and permanent faults. The α–count mechanism is so general that all the heuristic presented above can be easily obtained as α–count instances by properly

10 Hidden Markov Model
11 Triple Modular Redundancy

settings its internal parameters. The details about the α–count mechanisms are given hereafter.

The α–count mechanism

The α–count heuristic, proposed for the first time in [Bondavalli 97], was conceived for solving the problem of the discrimination between transient, intermittent and permanent faults, taking into account that "components should be kept in the system until the throughput benefit of keeping the faulty component on–line is offset by the greater probability of multiple (hence, catastrophic) faults" (citation from [Bondavalli 00]).

The faults and failure modes assumed for the monitored component are the following:

PERMANENT FAULTS: internal faults which lead the component to fail every time the component is activated.

INTERMITTENT FAULTS: internal faults which show a high occurrence rate and which eventually might turn into permanent faults.

TRANSIENT FAULTS: external faults which have an uncorrelated occurrence rate and should not determine the exclusion of the monitored component.

The α–count heuristic implements the "count–and–threshold" criteria, so the rationale behind the α–count is the following: many repeated errors within a little time window are easily evidence for a permanent or an intermittent fault; some isolated errors collected over time could be evidence for transient faults. α–count hence counts error signals collected over time, raising an alarm when the counter passes a predefined threshold, and decreasing the counter when not-error signals are collected.

We may assume that all fault–related events occur at discrete points in time and that two successive points in time differ by a (constant) time unit or step; we may assume also that the deviation detection mechanism evaluates at each step i whether the component is behaving correctly or not, issuing a boolean deviation detection information $J^{(i)}$ toward the diagnostic mechanism. We may assume $J^{(i)}$ is defined as follows:

$$J^{(i)} = \begin{cases} 0 & \text{if correct behavior is detected at step } i \\ 1 & \text{if incorrect behavior detected at step } i \end{cases} \tag{1.1}$$

α–count collects over time the signals coming from the deviation detection mechanism in a score variable α associated with the monitored component; at each step the score variable α is updated and compared with a threshold value α_T: if the current score value exceeds the given threshold α_T ($\alpha \geqslant \alpha_T$), the component is diagnosed as affected by a permanent or an intermittent fault, otherwise it is considered healthy or hit by a transient fault.

The score variable α is evaluated at each step i by using the following formula:

$$\begin{aligned} \alpha^{(0)} &= 0 \\ \alpha^{(i)} &= \begin{cases} \alpha^{(i-1)} \cdot K & \text{if } J^{(i)} = 0 \quad (0 \leqslant K \leqslant 1) \\ \alpha^{(i-1)} + 1 & \text{if } J^{(i)} = 1 \end{cases} \end{aligned} \tag{1.2}$$

where K is an internal parameter representing the ratio in which α is decreased after a step without error signals, thus setting the time window where memory of previous errors is retained.

Extended studies about the tuning of the internal parameters (K and α_T) are described in [Bondavalli 00], where the trade–off between promptness and accuracy is evaluated. Applications and variants (e.g. the double–threshold variant[12]) are described in [Bondavalli 04a, Serafini 07].

Practical applications of the α–count heuristic can be found also in the following works: [Powell 98], where α–count has been implemented in the GUARDS[13] architecture for assessing the extend of the damage in the individual channels of a redundant architecture; [Romano 02], where α–count is used in a COTS based replicated system to diagnose replica failures based on the result of some voting on the replica output results.

1.2.2 *A probabilistic approach for on–line diagnosis*

Diagnostic mechanisms following a probabilistic approach are tailored to evaluate over time the probability of the monitored component being faulty.

Probabilistic approaches for on–line diagnosis can be based on different underlying models: [Pizza 98] tackles optimal discrimination between transient and permanent faults applying the Bayesian inference to the observed events[14]; [Daidone 06] shares the same probabilistic view of [Pizza 98], but accounts for higher modularity and relies on a richer framework to solve diagnostic problems (more details about this probabilistic approach are described in section 1.2.2); [Nickelsen 09] applies Bayesian Networks (BN) for probabilistic end–node driven fault diagnosis of TCP end–to–end connections both in the wireless and wired domain.

The Bayesian inference

The core idea behind the probabilistic approach to diagnosis presented in [Pizza 98] is to use the Bayesian inference, which is expressed as follows: let us suppose to have a conjecture x about which there is uncertainty, and let $p(x)$

12 The double–threshold mechanism temporarily excludes the monitored component after the first threshold is exceeded, giving an opportunity to be reintegrated; the monitored component is definitely excluded as soon as the second threshold is exceeded.

13 Generic Upgradable Architecture for Real-time Dependable Systems

14 This approach describes how the assessed probability that a component is permanently faulty varies with observed symptoms, taking into account the coverage of the deviation detection mechanism.

be the degree of belief in conjecture x being true; after observing some new relevant evidence e it is desirable to update the belief $p(x)$ in a rational way. Both the evidence e and the conjecture x are described as events, that is subsets of the set of all possible outcomes of some experiment.

The similarity between the diagnosis problem and the Bayesian inference is based on mapping conjecture x with the fact that the monitored component is affected by a certain fault (listed in the fault assumptions), and mapping evidence e with the result of a deviation (error) detection information. The diagnostic mechanism is going hence to evaluate the probability $p(x)$ of the monitored component being affected by a certain fault (conjecture x), updating this probability when new detection information (evidence e) comes from the deviation detection mechanism. The diagnostic mechanism then exploits the evaluated probabilities in order to judge whether a proper alarm has to be raised, triggering this way the reconfiguration mechanism.

Diagnosis and the Bayesian inference

The Bayesian inference is exploited for diagnostic reasoning as described hereafter. The Bayes' theorem states the following:

$$p(x|e) = \frac{p(e|x) \cdot p(x)}{p(e)}. \tag{1.3}$$

The elements involved in equation 1.3 can be interpreted as follows:

$p(x|e)$ is the degree of belief in conjecture x after observing the new evidence e (let's call it "posterior" probability of conjecture x).

Reasoning in diagnostic terms, $p(x|e)$ is the probability that the component is affected by a certain fault (information related with conjecture x) given that a certain deviation detection information was collected (evidence e).

$p(x)$ is the degree of belief in conjecture x before observing the new evidence e (let's call it "prior" probability of conjecture x).

Reasoning in diagnostic terms, $p(x)$ is the probability that the component was affected by a certain fault (conjecture x) before collecting the last deviation detection information (evidence e).

$p(e|x)$ is the probability of observing some evidence e, given that the conjecture x is true.

Reasoning in diagnostic terms, $p(e|x)$ is the probability of the deviation detection mechanism emitting some detection information (evidence e), given that the monitored component is affected by a certain fault (conjecture x); this in some sense related to the imperfection of the deviation detection mechanism.

We may assume that C is the universe of all the possible conjectures c (e.g. all the faults listed in the fault model); the probability of conjecture x being true given that evidence e was observed can be updated exploiting formula 1.3 in the following way:

$$p_{\text{posterior}}(x|e) = \frac{p_{\text{prior}}(x) \cdot p(e|x)}{\sum_{c \in C} p_{\text{prior}}(c) \cdot p(e|c)} \tag{1.4}$$

We may now assume that all fault–related events occur at discrete points in time and that two successive points in time differ by a (constant) time unit or step. Moreover we may assume that the deviation detection mechanism evaluates at each step whether the component is behaving correctly or not, emitting a deviation detection information toward the diagnostic mechanism.

It is worth noting that the prior probability of a conjecture c at step i can be obtained as the posterior probability of the same conjecture c at the previous step $i-1$, adjusted based on the evidence observed in the meanwhile. The probability of the monitored component being affected by a certain fault at step i is hence obtained based both on the probability values evaluated at round $i-1$ and the deviation detection information collected in the meanwhile:

$$p_i(x) = p(x|e) = \frac{p_{i-1}(x) \cdot p(e|x)}{\sum_{c \in C} p_{i-1}(c) \cdot p(e|c)} \tag{1.5}$$

If the fault model of the component lists n faults, then formula 1.5 can be rewritten in order to deal with an n-component vector $\vec{p}_i$ formed by the single values $p_i(x)$ at varying x; this way the formula describes how the probability of the monitored component being failed can be updated over time based on the deviation detection information collected over time.

The diagnostic mechanism based on HMM

This section presents the diagnostic mechanism based on Hidden Markov Models (HMM) as it was originally proposed in [Daidone 06], giving some details about the rationale behind the mechanism and showing some features of the mechanism itself. Some extensions to the diagnostic mechanism will be proposed in this work and will be described in section 2.4.

This section refers to the diagnosis problem and its actors by using general terms, like "monitored component", "deviation detection mechanism", "fault", without entering into many details, because the diagnostic mechanism is general and applicable to several problems with different specifications (e.g. diagnosis of an hardware fault or a malicious intrusion).

The use of the HMM formalism allows both the definition of a framework through which the diagnosis problem can be approached, and the definition of a diagnostic mechanism. The above mentioned framework allows to tune the diagnostic mechanism on the base of the peculiarities of the system in which is implemented (basically the fault model of the monitored component) and the peculiarities of the deviation detection mechanism (e.g. probabilities of false positive and false negatives).

The diagnostic mechanism based on HMM follows the probabilistic approach presented in section 1.2.2, taking advantage of the peculiarities of the hidden Markov models in order to give an intuitive but formal way of solving the diagnosis problem. The diagnostic mechanism evaluates the probability of the monitored component being faulty, based on the observation of the results of a deviation (error) detection mechanism which has incomplete coverage and accuracy. The diagnostic mechanism is flexible enough to take into account the uncertainty of both observed events and observers, so that it can also be used to diagnose malicious attacks: literature shows e.g. that the HMM theory is used to implement an anomaly-based intrusion detection system (e.g. [Jecheva 06]).

In order to better comprehend the rationale behind the diagnostic framework based on HMM, some basic information about the formalism is presented hereafter.

What is an HMM

A hidden Markov model is a formalism able to represent probability distributions over sequences of observations; an HMM is basically a Markov chain whose state is "not observable" (the state is indeed "hidden") which emits "observable" symbols depending on a probabilistic function of the state. A good tutorial about HMM and its use can be found in [Rabiner 90].

Let Ω be the finite discrete set of states and Σ be the finite discrete set of observable symbols; an HMM M can be expressed by the following quintuple:

$$M = (\Omega, A, \vec{\pi}(1), \Sigma, B) \tag{1.6}$$

The elements of the quintuple are detailed hereafter:

Ω, A and $\vec{\pi}(1)$ are the basic elements describing a first–order[15] time–homogeneous[16] DTMC[17] [Bolch 05], that is:

- the discrete set of (hidden) *states* $\Omega = \{\omega_1, \dots, \omega_n\}$;
- the $n \times n$ *state transition probability matrix* A, where $A(i,j) = a_{ij}$ is the conditional probability that the model moves to state ω_j given that it is in state ω_i at time t;
- the *initial state probability vector* $\vec{\pi}(1)$.

$\Sigma = \{\sigma_1, \dots, \sigma_m\}$ is the set of distinct (observable) *symbols* emitted by the HMM (the so–called *alphabet* of the model).

15 A first–order Markov chain is a chain in which the state at time t depends only state at time $t-1$ only.
16 A time–homogeneous Markov chain is a chain where the over–time behavior does not depend on the actual value of time t.
17 Discrete Time Markov Chain

B is the $n \times m$ *observable symbol probability matrix*, where the element $B(i,k) = b_i(\sigma_k)$ is the conditional probability that the model emits the symbol σ_k at time t given that the model is in state ω_i.

It is worth noticing that the probability of emitting σ_k at time t depends only on the state at time t (which in turn depends on the state at time $t-1$) and not on the symbol emitted at time $t-1$.

Assumptions

The description of the diagnostic mechanism based on HMM bases on the assumptions described hereafter (some of them are made only for simplifying the description); section 2.4 will describe how some of the following assumptions can be lightened or even relaxed at all. The following assumptions are made:

1. All fault–related events occur at discrete points in time and two successive points in time differ by a (constant) time unit or step.
2. There is only one deviation detection mechanism, which evaluates whether the component is behaving correctly or not, issuing a deviation detection information toward the diagnostic mechanism at each step.
3. The fault model for the monitored component is known and does not change over time; this means that the system designer knows both which faults affect the monitored component and the probability value that a certain fault can activate.
4. The characteristics of the deviation detection mechanism (completeness and accuracy) are known, so that the system designer knows which is the probability of the deviation detection mechanism detecting (or not) a fault. This holds for each fault in the fault model.

Setting up the diagnostic mechanism

The fault model for the monitored component is used to initialize the DTMC underlying the HMM, meaning that:

- The faults listed in the fault model are associated with different states in set Ω; usually one state is associated with the "no active fault" situation, sometimes some states are defined to be associated with particular combinations of faults.
- The probability values related to the occurrence of faults are used to initialize both the initial state probability vector $\vec{\pi}(1)$ and the state transition probability matrix A.

The characteristics of the deviation detection mechanism are used to initialize both set Σ and matrix B; in order to take into account the possible imperfections of the detection mechanism (e.g. the occurrence of false positives and false negatives) matrix E and G are introduced. Σ, E, B and G are initialized as follows:

1. Each error message that can be raised by the deviation detection mechanism is linked with a different symbol in Σ;
2. The probability values of the monitored component showing errors given that it is faulty (for each fault in the fault model and for each error message in Σ) are used to initialize the elements of matrix E; please, note that matrix E is similar to the observable symbol probability matrix B, but it is not the same: the difference is that E is related to an ideal deviation detection mechanism, B to the real one.
3. If the deviation detection mechanism were perfect, than it would raise an error signal exactly when the monitored component shows an error; in this case the probability of raising a certain error message given that a certain fault is active in the monitored component is the same as the probability of the component showing that error when affected by that fault, and hence $B = E$.

   When this is not the case, that is when the deviation detection mechanisms is not an ideal one, a square matrix G called *translation probability matrix* is used: rows and columns of the G matrix are indexed with symbols in Σ so that the generic element $G[i,j] = g_{i,j}$ corresponds to the probability that the ideal emission of symbol i is translated in the actual emission of symbol j. If the deviation detection mechanism is ideal, the G matrix is the identity matrix. The observable symbol probability matrix B is obtained as the row–column product between E and G: $B = EG$.

Using the diagnostic mechanism

The basic information used by the over–time diagnostic mechanism in order to take decisions and trigger alarms is the state probability vector $\vec{f}(t) = (f_{\omega_1}(t) \ \ldots \ f_{\omega_n}(t))^T$, which contains the probability values of the monitored component being affected by each one of the faults listed in the fault model.

The state probability vector $\vec{f}(t)$ is evaluated at each step based on the state probability vector $\vec{f}(t-1)$ evaluated at the previous step and the deviation detection symbol σ_t observed at time t; $\vec{f}(t)$ is defined by the following formula[18]:

$$\vec{f}(t) = \begin{cases} \frac{1}{s_1} \mathrm{diag}\left(B^{\sigma_1}\right) \vec{\pi}(1) & \text{if } t = 1 \\ \frac{1}{s_t} \mathrm{diag}\left(B^{\sigma_t}\right) A^T \vec{f}(t-1) & \text{if } t > 1 \end{cases} \tag{1.7}$$

The computational cost for evaluating $\vec{f}(t)$ at each step is $O\left(n^2\right)$.

The over–time diagnostic mechanism uses the probability values in $\vec{f}(t)$ to take decisions about the alarms to trigger (if any); several criteria can be used for taking decisions, let's give some ideas in the following:

18 Please note the following:s_t is a scaling factor used to normalize $\vec{f}(t)$, $\mathrm{diag}(\vec{v})$ is a diagonal matrix built from vector $\vec{v}$, and B^{σ} is the column of matrix B corresponding to symbol σ.

1. A threshold vector $\vec{d} = (d_1 \ \dots \ d_n)^T$ is defined, where $d_i \in [0, 1]$ represents the threshold probability for the corresponding value $f_{\omega_i}(t)$ in $\vec{f}(t)$: d_i is the threshold for the probability of the monitored component being affected by the fault associated with state ω_i. Some thresholds are upper thresholds, meaning that the alarm has to be raised when $f_{\omega_i}(t) > d_i$, others are lower ones (e.g. the threshold related with the "no fault" state). In case (at least) one of the thresholds is violated, the diagnostic mechanism raises a specific alarm toward the reconfiguration mechanism.

2. Ω is partitioned in subsets and the above idea of the threshold vector is applied on subsets of Ω instead on single elements ω_i; this is the typical case in discriminating permanent and intermittent faults against transient faults, where the states in Ω are partitioned in two sets: {No-Fault, Transient} and {Intermittent, Permanent} and a threshold value needs to be set on the latter partition only.

Of course, variations are possible in a number of directions; in general, the setting of the threshold criterion depends on the requirements of the specific application using the services delivered by the monitored component.

Advantages and disadvantages

The diagnostic framework based on HMM is very general, as underlying models are not bound to particular components or fault detection mechanisms; furthermore, it takes into account all the aspects involved in component's state diagnosis, keeping them separated in modular way. The high accuracy of the mechanism is demonstrable through the comparison with the existing optimal solution relying on the Bayesian inference theory.

The formal approach beside the diagnostic framework can also be a support at design time for studying different combinations of fault models and for evaluating the impact of different deviation detection mechanisms; it can also be used to tune the internal parameters of heuristic techniques that could be implemented in place of a probabilistic mechanism due to lack of computational resources.

1.3 NEW TOOLS

Diagnostic mechanisms used for traditional systems are chosen and tuned based on static assumptions about system behavior, fault model and detection mechanism; all those basic assumptions derive from contexts where the "strange" component behavior can be defined a–priori and assumed due to faults in the component itself. Nowadays we need to enlarge those context and consider more dynamic scenarios, where, for example, failure definitions or system specifications change over–time.

Old diagnostic machineries are not more good for the job: given the amount of indicators to observe they become ineffective; another element is the need for information correlation, which the old mechanisms do not support. It is clear that the new scenarios require new diagnostic tools, because the information source for modern diagnosis is going to be monitoring instead of error detection.

The following problems arise:

- Which event to collect, given that failure definitions and system specifications can change over–time.
- How to define streams of relevant data, given the amount of data that can be monitored.
- Which inference rules to apply on streams of data in order to perform diagnosis.

Some ideas about how to solve these problems will be given in chapter 2, by making combined use of some existing tools (on–line monitoring and simple–event–correlation) which will be presented in the following sections.

1.3.1 *On–line monitoring*

On–line monitoring aims at the observation of system events, from several perspectives (faults, attacks, vulnerabilities), while the system is in operation. Several proposals exist where monitoring tools and facilities are proposed with slightly different aims and capabilities. Some examples of automatic on–line monitoring tools are the following:

- SWATCH (Simple WATCHer.) [Hansen 93]: it generates alerts based of the recognition of specific patterns while on–line scanning log files; its main drawback is that it doesn't support means to correlate recognized events. Swatch is written in PERL.
- Logsurfer [Thompson 04]: it is based on SWATCH, but it offers some advanced features which are not supported by SWATCH: for example, it can group related log entries together and it can dynamically change its rules based on events or time. Logsurfer is written in C: this makes it extremely efficient from a computational point of view.
- LoGS [Prewett 04]: LoGS'rule set is dynamic and rules are written in Lisp (as the tool itself): this provides more flexibility in designing rules than other tools, but it requires programming experience on the part of the rule designers. LoGS is still immature: a beta release is available only.
- SEC (Simple Event Correlator) [Vaarandi 02, Rouillard 04]: it recognizes specific patterns based on predefined rules; SEC is able to correlate observed events and to trigger specific alarms. SEC is written in PERL.

Monitoring mechanisms in critical infrastructures are required to filter the events observed so to understand the nature of errors occurring in the system, with respect to multiple classes of faults, and so feeding the diagnostic facilities.

1.3.2 *Diagnosis based on simple event correlation*

This section presents an on–line diagnostic mechanism based on the use of simple event correlation. The monitoring information is extracted over–time reading system log files, filtering all the available information in order to identify only information relevant for monitoring purposes (the so called "events"); the diagnostic engine then processes over–time the flow of the observed events, identifying alarming situations both from the observation of alarming events (events that directly notify an alarming condition) and from the correlation of specific simple not–alarming events. The goal is in fact to recognize not only malfunctions on top of self–evident error events, but also alarming situations on top of observed (possibly not alarming) events.

The proposed approach is general enough to be used for the identification of different kinds of alarming situations, e.g. hardware faults (e.g. a "stuck at" error message), software faults (e.g. an exception message) or malicious attacks (e.g. several messages about a failed attempt to login as root).

The diagnostic approach

Log files contain a lot of information related to both functional and not functional actions occurring in the system; they contain information related to several components in the system, possibly components belonging to different architectural levels, and all this information is mixed together. Among all this information, there is some information that is relevant for the diagnosis process: for sure there are self–evident errors notifications, but also other not alarming information could be useful; sometimes in fact error notifications are hidden among various status events indicating normal operations or borderline situations, which, considered altogether, are alarming symptoms. Log files are sometimes the only way to monitor COTS components, which cannot be modified to incorporate specific probes and hence need to be monitored as black–boxes.

Automatic tools are needed to scan on the fly log files, in order to:

1. Filter information and identify the "events" relevant for the diagnosis process: events are both self–evident deviation detection messages and system status events which could be symptoms of malfunctions.

2. Correlate the filtered events in order to identify sets of not–alarming events which observed altogether are signals for higher–level problems.

The rationale behind the event correlation engine is a set of rules which define how basic events are correlated in composite events. Event correlation is a good opportunity to exploit the occurrence of different symptoms as a "prediction" for a certain fault to generate a failure.

The following phases need to be performed at design time in order to set–up the diagnostic mechanism based on simple event correlation:

1. *Event categorization*: events are classified in a hierarchical set of event categories.

   The top level category usually corresponds to a discrimination between "relevant" and "not relevant" events, where the "relevance" is related to the identification of faults. Relevant events should include, in addition to critical events, also "not critical" events related to some data measured in the system or to periodic system actions; these "not critical" events could be used to identify system malfunctions which are not signaled by explicit messages.

2. *Event filtering* ("optional" step): events are filtered in order to remove redundant information and reduce the amount of information to be processed by the diagnostic engine (mainly for performance reasons).

   An ideal filter should have the maximum compression rate and no loss in terms of semantics of relevant information, but the issue here is to discriminate which events are redundant and which information is relevant. Typically redundancy is both on the time scale (e.g. repeated error messages related to the same fault) and on the space scale (e.g. more than one deviation detection mechanism detects the same error on the same monitored component).

3. *Association rule definition*: rules are defined to correlate the events identified (and possibly filtered) in the previous phases; mining rules are used both to execute matches among observed events and to generate the proper diagnostic judgments/alarms.

   The knowledge of the diagnosed system lets the system designer to create matching rules; examples of matching criteria are the following:

   a) In the time domain: two (or more) events are observed within a temporal window, and hence they are correlated; based on the specific requirements, the temporal window is defined either as a fixed length window (e.g. to search for a missing event) or by a pair of begin/end events.

   b) For causal reasons: two (or more) events are observed in a given order and hence they are correlated; the reason for the correlation is the event causality: the first observed event is the cause for the next one (which is the effect).

   c) For localization reasons: two (or more) events, generated by the same source (system component), are observed and hence correlated as symptoms for the same malfunction (e.g. repeated application errors raised by the same process could be interpreted as manifestation of software aging).

The identification of the mining rules can be partially automated by using specific techniques (something similar to the "association rule learning" in data mining), but this is out of the scope of this study.

Using the event correlation for diagnosis purposes

The monitoring tool that can be used as a support for modern diagnosis (as it will be proposed in chapter2) is SEC. SEC is an open–source rule–based event correlation tool, created in an academic context, and has interesting characteristics in terms of portability (is PERL based) and updatability (the rule set can be updated on the fly). Details about the mechanism itself and its use in conjunction with system log files can be found in [Rouillard 04].

SEC can be seen as a complex, context–aware filter which selects and correlates relevant detection information based on matching rules defined using regular expressions; rather complex matching patterns can be defined in a compact way, that would otherwise be quite awkward to express. SEC self–learning capabilities are masked, but existing: rules are defined in specific configuration files (text format) called *rule files*, which can be refreshed at run–time keeping the status of the ongoing correlations. Rule files are refreshed when the SEC process is requested to perform a "soft restart" from the hosting operating system through specific inter–process signals (`SIGABRT` or `SIGTERM`). In particular, a soft restart consists of the following steps:

1. Rule files are reopened (new files can be opened too).
2. Event correlation operations started from rule files that have been modified or removed after the previous configuration load are canceled.
3. Other operations and other event correlation entities (contexts, variables, child processes, etc.) remain intact.

SEC rule sets are defined based on nine rule types, which can be classified into two groups: *basic rules*, which perform actions and do not start an active correlation, and *complex rules*, which start a multi–part operation that lasts for some time after the initial event and which perform the actual correlation.

Basic rules are the following:

SUPPRESS: this rule suppresses the matching input event, so that the event is not matched by other following rules.

SINGLE: this rule executes an action list l if the input event is matched.

CALENDAR: this rule executes an action list l at specific times.

Complex rules are the following:

singlewithscript: this is an extension of the "single" rule, so it executes an action list l if the input event is matched and -this is the difference- if also an external script returns a certain output value.

singlewithsuppress: this is an extension of the "single" rule, so it executes an action list l if the input event is matched, but -this is the difference- it ignores the following matching events for the next t seconds.

pair: this rule executes an action list l_1 if the input event is matched, then it ignores the following matching events until some other input event is matched; on the matching of the second input event, another action list l_2 is executed.

pairwithwindow: this rule executes an action list l_1 if the input event is matched, then it waits for t seconds for a second input event to be matched: if the second input event is not observed within the time window, it executes an action list l_2, otherwise (the second input event arrived on time) it executes another action list l_3.

singlewiththreshold: this rule counts how many times the input event is matched within a time window of t seconds; if a given threshold is exceeded within the given time window, it executes an action list l and ignores all matching events during the rest of the time window.

singlewith2thresholds: this rule counts how many times the input event is matched within a first time window (duration t_1 seconds); if a given threshold is exceeded within the given time window, it executes an action list l_1. Then it restarts counting how many times the input event is matched within a second time window (duration t_2 seconds); if the counter is below a second threshold, it executes another action list l_2.

SEC reads streams of information about events occurring in the system and triggers the execution of both internal actions and external processes (e.g. shell commands); input is read on the fly from both log files and system pipes on a line–by–line fashion. Rules are used in particular to:

- Create and delete contexts, which are used to discriminate whether a given rule has to be applied or not in a certain scenario;
- Associate collected events with a single context and report collected events at a later time;
- Generate new (intermediate) events that will be input for other rules;
- Reset correlation operations that have been started by other rules.

Combining several rules and contexts together, complex event correlation schemes are defined.

2 A DIAGNOSIS FRAMEWORK

This chapter proposes both a diagnosis framework and a conceptual schema which together aim to cope with the growing complexity of the problem of diagnosis in modern critical infrastructures. The objective is to perform monitoring and diagnosis activities, trying to correlate monitored information for diagnosis purposes.

The diagnosis framework suggests that system diagnosis is performed at run-time in a distributed way, taking into account both the local and the global point of view: each node has to diagnose both itself (local diagnosis) and other nodes (private diagnosis) based on the local perception of their behavior; in certain moments of time (e.g. when a serious malfunction is detected inside a set of collaborating nodes) distributed diagnosis is performed, in order to reach consensus about the healthy status of that part of the system (possibly of the entire system).

The conceptual schema aims to make advanced use of available monitoring tools by means of a new variety of on–line diagnostic mechanisms. The goal is to build an evolutionary collection of detectable effects of the problems affecting the system (e.g. faults, intrusions), to enable early recognition of intricate space/time malfunction patterns.

Some problems have to be considered: i) the existence of wrong detections, due both to the imperfection of detection mechanisms themselves and to the deviated perception of remote activities (e.g. because of communication problems); ii) the presence of malicious faults, aimed to distort the perception of the various system parts (e.g. a node tries to persuade other nodes that a certain server is congested in order to be the sole to gain its services); iii) the cost of distributed diagnosis (despite of the use of an authentication mechanism).

2.1 REQUIREMENTS AND SPECIFICATIONS

The modern critical infrastructures considered in this work have specifications and requirements[1] that pose some issues from the viewpoint of assuring system resilience. Let's present the main diagnostic issues by recalling the main infrastructure specifications in the following list:

- Infrastructures are made by connecting previously stand–alone systems, usually developed from proprietary architectures, where ad–hoc solutions

1 The discussion about current status and future trends of critical infrastructures can be found in the introduction.

were chosen and several components were developed independently; ad–hoc mechanisms for detection, diagnosis and reconfiguration were used, which now need to be coordinated. The original mechanisms were designed in strict dependence with the proprietary stand–alone system only, not considering the following integration in some infrastructures, so those mechanisms could now act "against" the infrastructure's interests.

- Sub–systems composing the infrastructure were not designed to be widely distributed and remotely accessed, and they do not cover security issues; the interactions among those sub–systems need to be treated in some way, adding the necessary coordination support for the management both of fault tolerance and security (treating e.g. both hardware and malicious faults).
- Large–grained components are used, which have a lot interactions among them and their sub–components; it is hence difficult to link a single error appearing within the infrastructure with a well focused fault in a specific bounded component. Moreover, the goodness of a component could be related to the quality of the service provided, rather than to the absence of faults, affecting this way diagnostic judgments and reconfiguration strategies.
- The environment in which diagnosis is performed is heterogeneous; many different entities coexist in the infrastructure: this requires each of them having specific solutions for assuring resilience.
- The infrastructure is distributed by its very nature and hence communication and coordination problems need to be solved.

On–line diagnosis

Diagnosis in critical infrastructures is required to assure the resilience of the infrastructure itself, performing on–line activities tailored to assess the status of the monitored components and the extent of the faults which can possibly affect them.

Fine– or large–grained?

Diagnosis activity has to be performed at different granularity levels (FRUs[2]), depending on the controllability of the monitored components (e.g. when dealing with COTS and legacy sub–systems) and on the cost/efficacy ratio of the detection–diagnosis–reconfiguration operations. On one side, fine–grained diagnosis is very helpful, since it allows replacement of smaller parts of the system, avoiding wasting still useful subparts of the components under diagnosis. On the other side, fine–grained diagnosis incurs in higher costs from the point of view of setting up diagnosis activities. Opposite trends are instead shown by a large–grained approach.

Information correlation

In large, well designed and tested systems, with hardware components far

2 Fault Replacement Units

from their wear out age, crude faults or malfunctions, easily and rapidly recognizable as such, tend to be rare events; while subtle borderline conditions may still occur, whose very presence is difficult to detect, not to mention accurate recognition and treatment. Another difficulty derives from the observation that, in the less–than–simple systems, a binary (faulty/not–faulty, go/not–go) schematization of the error behavior is insufficient and possibly counter–effective on the availability of the component.

More often, in a mature system non–fatal malfunctions occur which, although far from downing the entire affected (sub)system, nevertheless require corrective action. Also from this point of view, then, finer recognition of borderline situations is needed, to enable flexible reaction.

Even if a binary decision scheme is considered good enough, an approach more knowledgeable than just waiting for a single, unambiguous error signal would have a positive impact on system dependability.

2.2 THE PROPOSED DIAGNOSIS FRAMEWORK

A distributed approach

Given the geographical extension of critical infrastructures it is not practical to have a centralized diagnostic entity that has to gather and analyze all the information about the detected deviations in order to diagnose the system; this kind of centralized state diagnosis should be ultra–reliable and communication links to all the parts of the system should be guaranteed. The fact that the critical infrastructure is formed by several interconnected sub–systems leads the infrastructure to be logically decomposable in a set of interconnected islands (nodes), each one endowed with its own monitoring and diagnostic tools. Therefore methods for distributed diagnosis (and reconfiguration) are mandatory, where each node decides independently about the system (e.g. which are the healthy nodes and which the faulty ones). Then, reasoning top–down in a hierarchical way, we can apply the same rationale (if necessary) within each system node (island).

Local vs. global classification of fault effects

Considering a distributed system comprised by completely connected nodes, the hybrid fault–effect model [Walter 97] can be assumed, so that all fault classification is based on a local classification of fault effects (to the extent permitted by the deviation detection mechanism of the node itself) and on a global classification, thus developing a global opinion on the fault effect. Diagnosis is thus performed using a two–phase approach on a concurrent, on–line and continual basis:

1. Local detection and diagnosis, based on the local perception of the node about the faulty status of other nodes.

2. Global information collection and global diagnosis, obtained through exchange of local diagnosis.

Since each node may have a different perception of the errors created by other nodes, each node has some private values (the results of local diagnosis on remote nodes) and the goal is to ensure consistent information exchange and agreement against Byzantine behavior. The above approach can hence be extended by considering the following three diagnostic scenarios:

LOCAL DIAGNOSIS: the node judges the status of its internal components/services, based on information available in the node itself.

PRIVATE DIAGNOSIS: the node judges the local perception of remote nodes/services, based on information collected by the node itself.

DISTRIBUTED DIAGNOSIS: the node participates and contributes to the building of a common view about the status of remote services in the system.

Diagnosis strategy

Based on the diagnosis scenario identified above, which will be detailed in the following sections, the general diagnosis strategy turns out to be the following:

1. Every node diagnoses itself over time, in order to judge the healthy/faulty status of its resources/services; this step determines the local status of the node, which is used primarily for local reconfiguration aims.

2. Every node builds a private judgment about the other nodes and the services they provide, based on the perceived behavior observed through direct functional relationships with them or through specific challenge–response tests; this step produces the so–called *perceived status* of a certain node, which is used to discriminate whether the remote node is trustable or not.

3. Every node, when asked for, declares its local status (everything, only relevant parts or a signature of them); since nodes are not completely trusted by other nodes, they do not completely trust the *declared status*.

4. When necessary, (groups of) nodes exchange among them their perceived status of a certain node, in order to reach an agreement about the trustworthiness of that node. This step is affected by the fact that the perceived status and the declared status of a certain node could be conflicting (e.g. because of communication problems or because of deliberate and malicious causes); moreover, two different nodes could perceive the same remote node in different ways.

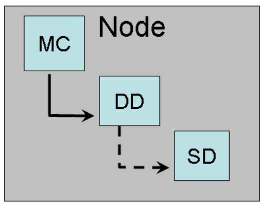
Hybrid architecture

The critical infrastructure is distributed in its very nature, and it is hence exposed to communication and coordination problems, as well as to those caused by hardware or operating system; it is not possible to manage these problems at the component level, but it is necessary to do that at a higher architectural level. From an architectural point of view, it seems that the more suitable way to integrate diagnostic mechanisms in the system is to integrate them at middleware level and take advantage of hybrid architectures in order to provide the basic services with certain guarantees.

Hybrid architectures are system architectures where distinct parts of the system have different properties and are based on different sets of assumptions (including e.g. faults and synchrony properties). The literature about distributed systems, in particular referring to timing properties of systems, includes an hybrid architectural paradigm based on the so–called *wormholes* [Veríssimo 06a], which are special components having stronger properties than the other system components and that provide services (e.g. temporal references) with certain guarantees. System resilience can be enforced in design phases by using an hybrid architecture: relevant examples are the MAFTIA[3] architecture [Powell 03], the CRUTIAL architecture [Veríssimo 08] and the HIDENETS architecture [Casimiro 07].

2.2.1 *Local Diagnosis*

Local diagnosis aims to determine the local status of a node; local diagnosis is performed internally on local resources/services, so the monitored components, the deviation detection mechanisms and the diagnostic mechanism are all inside the node (as depicted in figure 2.1).



MC Monitored Component: “hidden” internal state, “observable” external behavior.

DD Deviation Detection mechanism: imperfect coverage and accuracy.

SD State–Diagnosis mechanism: judgment about MC internal state based on imperfect information.

$\rightarrow$ observation of external MC behavior.

$\dashrightarrow$ information about perceived MC behavior.

Figure 2.1: Architectural view of the local diagnosis scenario.

The accuracy of the diagnostic mechanism in the local scenario is affected primarily by the characteristics of the deviation detection mechanism. Deviation detection mechanisms, deeply studied in the literature of fault tolerance [Siewiorek 98], are characterized by a level of accuracy and a level of completeness (coverage); very often, neither the accuracy nor the completeness are 100%, leading deviation (error) messages to do not perfectly reflect the internal healthy state of the monitored components (false positives and false negatives

3 Malicious–and Accidental–Fault Tolerance for Internet Applications

exist). The diagnostic mechanism has therefore to filter error (deviation detection) messages in order to judge whether the monitored component is still beneficial for the node or it is better to signal it as “suspect” (possibly triggering some reconfiguration/maintenance on it).

The diagnosis of a single component can be performed by implementing heuristic or probabilistic mechanisms as those described in section 1.2. If several monitored components at different architectural levels need to be diagnosed at the same time, then the combined use of both traditional diagnostic mechanisms and traditional correlation information tools (described in section 1.3.2) is recommended: all the available monitored information can be processed over time in order to recognize symptomatic situations which generate errors. The full details about the rationale behind the correlation of monitored information for diagnosis purposes will be proposed in section 2.3.

In those cases in which the hybrid architecture is implemented, the diagnostic mechanism can exploit deviation detection information collected from the services running in the wormhole as trustable detection information.

2.2.2 *Private Diagnosis*

Private diagnosis is performed by a node on a remote node (or service), based on its perception of the behavior of the remote node (or quality of the services provided); the connection between the local node and the remote node is the first source of uncertainty about the collected deviation information. The detection of the remote service/node should be done exploiting both the functional messages exchanged between the two nodes (as shown in [Walter 97]) and some tests explicitly designed for the job. The detection based on observation of functional behavior requires that a functional relation holds between the two involved entities in order to collect information; explicit test can be instead triggered on-demand or on a periodic base.

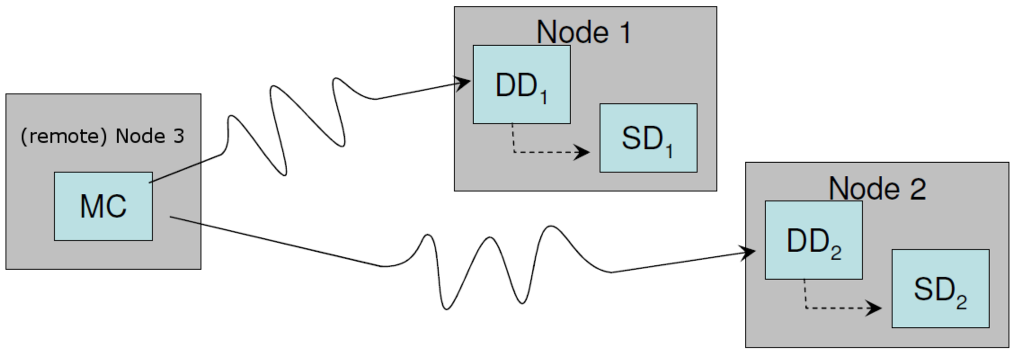


Figure 2.2: Architectural view of the private diagnosis scenario where *Node 1* and *Node 2* are diagnosing the remote *Node 3*.

Figure 2.2 shows the private diagnosis framework, where two nodes (*Node 1* and *Node 2*) are observing the remote *Node 3*; communication between the tester node (e.g. *Node 1*) and the tested node (e.g. *Node 2*) is an element that could (negatively) affect the perception of the remote node's services, leading to incorrect judgments. Two nodes could privately diagnose the same remote node in different ways, possibly in a conflicting way, due to diverse communication relationships!

What to detect

Examples of elements that can be observed on the behavior of a remote node are the following:

1. Errors on existing communications between the local and the remote node:
   - the tester node performs parity checks, checksums, message framing checks, range checks, sanity checks, comparison techniques on messages received from the remote node;
   - the tester node detects whether early/delayed messages come from the remote node.
2. Quality of services that the tester node requires from the remote node.
3. Ad–hoc tests requested by the tester node to the tested node; these tests are similar to "challenge-response" tests used in cryptography for authentication purposes. "Challenge-response" tests are useful in order to stimulate all the system parts of the tested unit, trying to excite latent faults (the observation of common message traffic is less effective for this issue).

Private diagnosis is used as long as the local node needs to use the service provided by the remote node and hence the private diagnosis results can affect the local node only; there are cases in which some nodes need to reach an agreement about the same remote node: in this cases, given the available and possibly conflicting information about the perception of the remote node, the private diagnosis is not enough and the distributed diagnosis needs to be performed.

2.2.3 *Distributed Diagnosis*

Distributed diagnosis is performed in a distributed way, among a set of collaborating nodes, in order to reach an agreement (Byzantine resilient) about the healthy status of a remote node; figure 2.3 shows the distributed diagnosis scenario in which some nodes (*Node 1*, . . . , *Node n*) have each different private perceptions of the same remote node and hence need to collectively reach an agreement about the remote node.

A consensus algorithm [Walter 97] ensures that the following properties are fulfilled:

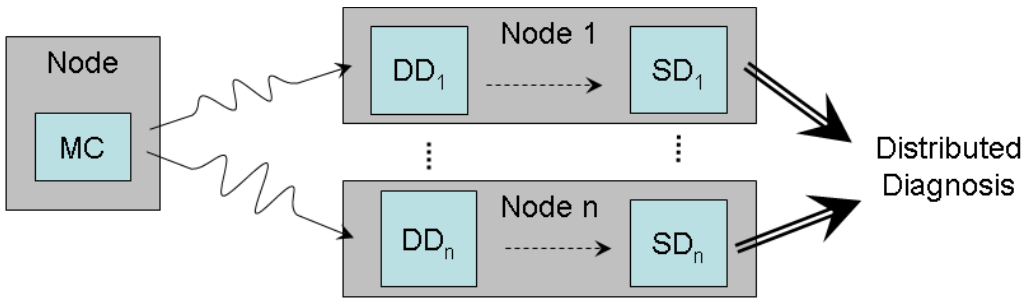


Figure 2.3: Architectural view of the distributed diagnosis scenario.

AGREEMENT: if *Node A* and *Node B* are non-faulty, then they agree on the value ascribed to any other node.

VALIDITY: if node *Node A* and *Node B* are non–faulty, then the value ascribed to *Node A* by *Node B* is indeed the private value of *Node A*.

In the general case, the necessary conditions to achieve consensus in spite of up to f arbitrarily faulty nodes are:

1. at least $3f + 1$ nodes in the system;
2. at least $f + 1$ rounds of message exchange.

Under the assumption of authenticated messages, which can be copied and forwarded but not altered without detection, the condition on the minimal number of nodes can be relaxed to $f + 2$ [Gong 95].

Distributed diagnosis is an interactive consistency problem where nodes, broadcasting their private diagnosis about the monitored component, want to reach an agreement about the system– (group–) level diagnosis of the monitored component (that is the status of a specific node). The above idea can be extended diagnosing the status of all the nodes in the system (group).

Deviation detection data about the specific monitored node is collected over time in the system (each node collects a part of that data): distributed diagnosis is a sort of data mining technique aimed to identify patterns related to faults.

Several existing protocols are available in order to implement distributed diagnosis [Barborak 93, Powell 98], but most of them require a lot of communication resources ([Martin 06] being the cheapest one in the common case); the first step in order to save resources is to try to limit the number of nodes involved in the distributed protocol. Since some activities not always have to involve the entire system, groups of nodes can be defined. The existence of groups of nodes raises the problem of the membership management (e.g. how to deal with group partitioning?), but every solution should be performed in a distributed way (group leadership should be avoided).

Distributed diagnosis should be performed periodically (the period should be clearly tuned), but some events could request for a specific distributed diagnosis run, e.g. the membership algorithm signals a new node entering in or exiting from the group, or some nodes locally detect a specific malfunction whose negative effects could not be limited to their local environment. The request for periodic diagnosis could lead to DoS attacks, so specific countermeasures need to be applied.

The agreement algorithms reach an agreement about one of the values proposed by at least one member of the group; in the case of distributed diagnosis, the values exchanged are the different diagnostic judgments. Since we have also private diagnostic judgments, the final agreed value could be something "in the middle": in this sense the agreement becomes an algorithm for "information fusion" (e.g. the final agreed value is a weighted mean where the weight derives from the private diagnosis of the proposer).

This proposal poses the following problems:

- How to assign the weight to the singe contributions;
- How to evaluate the Byzantinism of a node.

Cost

The costs of a distributed diagnosis run is defined in terms of i) time and ii) communications (how many message exchanges are necessary to complete diagnosis).

When

When it is proper to perform diagnosis inside a group of nodes:

1. Membership algorithm signals some node entering in or exiting from the group:
    - the group needs to know who is the new member,
    - the group needs to reorganize the group knowledge.
2. Some node locally detects a specific malfunction whose negative effects could not be limited to the local environment.
3. Periodically (the period has to be lower than the MTBF of nodes)
    - periodic diagnosis is demanded by the group leader (if any),
    - periodic diagnosis request could lead to DoS[4] attacks.

[Powell 99] presents an example of a distributed diagnosis algorithm in the scope of the GUARDS project; the algorithm is a distributed version of the heuristic α–count where some numeric values, being α–scores, are exchanged and voted among system nodes; the same basic criteria can be reused using probabilistic distributions instead of α–scores, defining this way a distributed probabilistic diagnostic mechanism.

4 Denial of Service

2.3 THE PROPOSED CONCEPTUAL SCHEMA

Processes and data sets for information correlation

We aim to identify symptomatic situations which require an intervention before an error fully develops; we aim also to find new error patterns deriving from changes of the system (e.g. evolution), of the environment, or even of the way the system is used.

The treatment of "simple" errors is already incorporated within sub–systems; we aim to analyze situations requiring more complex tools. We aim to recognize and catalog symptoms of several types coming from different sub–systems in order to be able to associate a name and a diagnosis to patterns of those symptoms (syndromes). Diagnosis is hence seen as the identification of a malfunction that can lead to the collected set of symptoms, which by themselves can be either already known or requiring to be classified as dangerous.

The monitoring activity on a certain system component is based on the observation of error signals occurring during the component lifetime; simple malfunctions, directly detected by component error detection mechanisms, are self evident error events. Each diagnostic approach can in fact be mapped on the framework described in section 1.1.1 (and depicted in figure 1.2): the monitored component, which is affected by faults (a fault model is assumed) and hence could behave incorrectly, is observed by a deviation detection mechanism, which periodically checks (with incomplete coverage and imperfect accuracy) whether the component behaves correctly or not. The check results produced by the deviation detection mechanism are then filtered by a state diagnosis mechanism, which is in charge of judging which fault (if any) is active in the monitored component, possibly rising appropriate alarm signals.

When diagnosis is performed on–line, streams of data on component behavior are collected and filtered over time; the filtering function could use an heuristic approach (e.g. the α–count mechanism presented in section 1.2.1) or a probabilistic approach (e.g. the diagnostic mechanism based on HMM presented in section 1.2.2).

When dealing with more than one component, or more than one deviation detection mechanism, the above diagnosis framework need to be instantiated accordingly. If the number and the complexity of monitored components grows up and the interdependencies among components behavior become larger, subtle malfunctions at component level can possibly give rise to erroneous manifestations elsewhere in the system, with patterns (set of symptoms) not immediately pointing to the actually faulty component. Moreover, system level improper conditions may arise from unexpected combinations of otherwise legal component behaviors, obviously signaled as correct events.

Poly–events

In order to recognize those situations, it is necessary to look not only for single events, but also for sets of correlated events that altogether lead to system malfunction; we will refer to those sets as "poly–events".

The conceptual schema describing the operations of monitoring and information correlation for diagnostic purposes is presented hereafter. We assume that the information about the nature and effects of relevant events is recorded in a number of event sets ({*SG*}, {*B*}, {*G*}) and processed by some processes (*Collector*, *Normalizer*, *Aggregator*, *Recognizer*). The event sets are the following:

{*SG*} is the repository of events occurred in the system, which are not yet associated to known poly–events; each event is kept in {*SG*} until an event–specific deadline (possibly infinite) is reached.

{*B*} is the set of "bad" poly–events, those which have been recognized as system malfunction syndromes; a *NYE* (Not Yet Established) flag is initially set for poly–events whose negative connotation is still to be confirmed.

{*G*} is the set of "good" poly–events which represent normal situations; for the sake of efficiency, in the implementation only "good" poly–events that include several events found in {*B*} need possibly to be stored.

The conceptual schema, depicted in figure 2.4, manages the monitored information by using the processes described hereafter.

Collector

A *Collector* process collects information streams about events occurring in the system (e.g. events could be read from application– or system–log files); each event is characterized by several attributes (which are application– or system–dependent), some of which are the following:

1. *Timestamp*, to have a temporal reference;
2. *Type*, to classify the event from a high level point of view;
3. *Severity*, to be able to give priority to severe events;
4. *Architectural level and localization*, to be able to correlate events based on localization;
5. *TTL* (Time To Live), to know how much time the event must be taken into account;
6. *Resettable* (or not), to know whether a subsequent event can cancel it or not.

The amount of events occurring in the system could be huge, so a pre–filtering function could be used to select the relevant events that the *Collector* should process. This step should at the same time i) cut (or compress) some events to reduce the number of processed event and speed–up the subsequent steps, and ii) do not cut the events that could be relevant for the subsequent steps; this trade–off is argument of current research (e.g. see [Liang 05]).

Normalizer

The format of the events collected by the *Collector* is source-dependent, so a

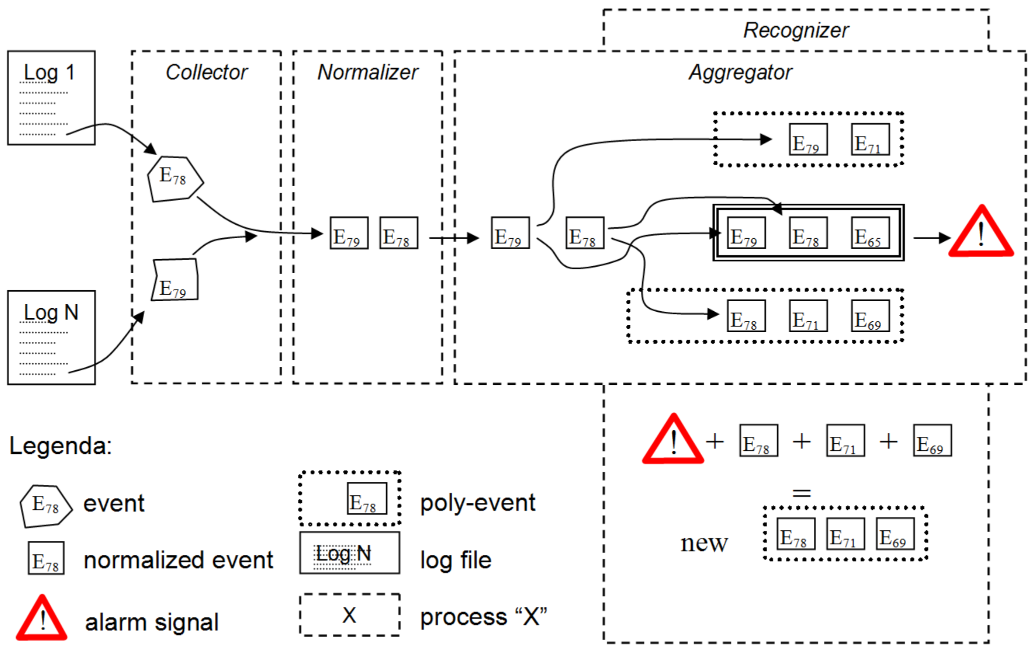


Figure 2.4: Conceptual schema of information correlation for diagnostic purposes.

Normalizer process works together with the *Collector* in order to translate events in records having a uniform format. For easiness of presentation, let's use the same word "event" for both the source-formatted event collected by the *Collector* and the corresponding normalized event (record) translated by the *Normalizer*. Normalized events are stored in the $\{SG\}$ set.

Aggregator

The next step is to search $\{SG\}$ for anomalous conditions. An *Aggregator* process searches $\{SG\}$ for sub–sets of events that partially of fully match poly–events in $\{G\}$ and $\{B\}$ (i.e. making use of previous knowledge). The search criteria is the following: upon the notification of an event e, the *Aggregator* first checks its severity, to ascertain if an immediate action has to be taken - in that case it rushes over the proper action request; next, event e is matched against a number of event sets:

1. Poly-events already open, waiting for further matches; add the event to all the open poly–events where a match is found: if a poly–event is now fully matched, the *Aggregator* generates the proper diagnostic result.

2. Poly-events already known as symptoms of malfunction, collected in $\{B\}$; a matching poly–event is opened.

3. Poly-events known as signs of normal operation, collected in $\{G\}$; a matching poly–event is opened.

Several matching criteria may be used; to start, events in $\{SG\}$ obviously exhibit correlation in time, to some degree. The *Aggregator* process picks up a tentative poly–event, and looks for other correlations among events, trying to consolidate a more significant poly–event out of it, i.e. it associates other events, possibly taken from other elements in $\{B\}$, according to several rules, to extract a variegate picture of the system behavior. The rules used to correlate event e with other events are the following:

1. *t-rule*: events occurred in the last t time units wrt event e;
2. *s-rule*: events occurred in a set S of physical enclosures (circuit board, subassembly, rack, room) in which event e has occurred;
3. *p-rule*: permanent events, always correlated until the system is reset;
4. *a-rule*: events occurred as part of, or alongside to, an activity α affected by event e.

A distance function is defined for each rule; the distance between two events is defined as a function (e.g. the RMS[5]) of the distances computed by each rule. The definition of the distance function is relevant to prevent the tendency of poly–events to grow without bounds, because small poly–events tend to be gobbled up by larger ones, while often small poly–events, possibly single events, may be very significant! The tentative poly–event P_t is aggregated to poly–event P_s if:

1. the number of events in P_t is lower than the number of events in P_s;
2. for each event e_t in P_t it is possible to find a corresponding event e_s in P_s such that the distance between e_t and e_s is lower than a given threshold.

If P_t cannot be aggregated to any pre–existent poly–event, it is set as a new element in $\{G\}$, with a confidence parameter set to "low".

If none of the above matches succeeds, the *Aggregator* appends e to a time-ordered sequence which is being built up; this sequence will be truncated upon the occurrence of a "trigger" event is reached (such as a bus error, an application crash, a network link down, an interval time expiring, a "max event count"). The list of triggers is initially populated with known adverse events, and can be updated by the *Aggregator*, whenever a new error condition is shaped up. The events belonging to the truncated sequence are boxed into a tentative poly–event set, which is then inserted in $\{B\}$ with the *NYE* flag set. The collection of a new sequence is started afterwards.

Recognizer

The *Recognizer* process is in charge of recognizing if a tentative poly–event, or a subset thereof, or even a superset thereof, is to be signaled to the error processing sub–system. It draws from previous knowledge of positively acknowledged malfunctions, but also from direct signals from higher levels, notifying out-of-spec behaviors possibly escaped to the monitoring sub–system. So, while the

5 Root Mean Square

Collector and the *Aggregator* operate in a strict bottom–up fashion, the *Recognizer* acts both bottom–up and top–down, helping in this way to fill detection gaps.

2.4 SOLUTIONS PROPOSED TO EXTEND THE DIAGNOSIS BASED ON HMM

This section proposes some extensions to the probabilistic diagnosis mechanism proposed in [Daidone 06] and described in section 1.2.2; the original contribution presented hereafter deals in particular with the assumptions described in section 1.2.2, explaining how they can be lightened or even relaxed at all.

Missing setup information

The diagnostic mechanism described in section 1.2.2 assumes that the following parameters are known in order to setup the mechanism:

1. Which faults can affect the monitored component; this information is necessary to define set Ω.
2. The probability values related to the occurrence of the assumed faults, in particular the status of the component at the beginning of its operational life; this information is necessary to initialize vector $\vec{\pi}(1)$ and to fill all the entries of matrix A.
3. Which error messages can be raised by the deviation detection mechanism; this information is necessary to define set Σ.
4. The probability values related to the emission of the deviation detection messages given the faults active in the monitored component; this information is necessary to fill all the entries of matrix B.

Whilst some of the above information is usually easy to find, e.g. which faults can affect the monitored component or which error messages can be observed, other can be tricky to find, e.g. the probability values related to the occurrence of faults or to the emission of deviation detection messages. Transposing this consideration in HMM terms, whilst Ω, Σ and $\vec{\pi}(1)$ can be easily identified, A and B can be problematic to setup.

Literature about HMM proposes a solution for setting up A and B, given that Ω, Σ and $\vec{\pi}(1)$ are known: the Baum–Welch algorithm [Rabiner 90]. The idea is to give the parameters an initial temporary value, then to use some "learning sequences" $S_1, \ldots, S_l$ (relevant sequences of observed symbols) to refine the model parameters A and B in order to maximize the probability of the model emitting (recognizing) those learning sequences. The best result is obtained when each learning sequence $S_i = \langle \sigma_1, \ldots, \sigma_k \rangle$ is provided with the corresponding state sequence $O_i = \langle \omega_1, \ldots, \omega_k \rangle$, so that the algorithm knows that symbol σ_1 was emitted while in state ω_1, etc. .

Thinking in diagnostic terms, this solution requires the existence of a prototype for both the monitored component and the deviation detection mechanism to generate the learning sequences. A learning sequence S_i is a sequence of deviation detection messages; the corresponding state sequence O_i is the sequence of healthy/faulty states the monitored component was while S_i was observed.

The solution provided by the Baum–Welch algorithm is more accurate if the following conditions hold: i) the initial temporary values are close to the real ones, ii) the number and relevance of learning sequences is high, and iii) the learning sequences are used with the corresponding state sequences.

Missing deviation detection messages

The diagnostic mechanism described in section 1.2.2 assumes that a deviation detection information (σ_t) is available at each time step, so that the state probability vector $\vec{f}(t)$ is evaluated using the following formula (which is the same as Formula 1.7 presented in section 1.2.2):

$$\vec{f}(t) = \begin{cases} \frac{1}{s_1} \mathrm{diag}\,(B^{\sigma_1})\,\vec{\pi}(1) & \text{if } t = 1 \\ \frac{1}{s_t} \mathrm{diag}\,(B^{\sigma_t})\,A^T \vec{f}(t-1) & \text{if } t > 1 \end{cases} \tag{2.1}$$

Formula 2.1 refreshes the probability vector $\vec{f}(t)$ at each step based on both the state transitions probability values (information enclosed in matrix A) and the probability values related to the emission of the observed symbol σ_t from each state in the model (information enclosed in matrix B). What if the observed symbol σ_t is missing?

An HMM is basically a DTMC with some extra features related to the observed symbols, so when the observed symbol is lacking we can think as we are dealing with a DTMC: we can update $\vec{f}(t)$ by using the transition state probability matrix only: $\vec{f}(t) = A^T \vec{f}(t-1)$. The new formula for updating $\vec{f}(t)$ at each step is hence the following:

$$\vec{f}(t) = \begin{cases} A^T \vec{\pi}(1) & \text{if } t = 1 \land \neg\exists\sigma_1 \\ \frac{1}{s_1} \mathrm{diag}\,(B^{\sigma_1})\,\vec{\pi}(1) & \text{if } t = 1 \land \exists\sigma_1 \\ A^T \vec{f}(t-1) & \text{if } t > 1 \land \neg\exists\sigma_1 \\ \frac{1}{s_t} \mathrm{diag}\,(B^{\sigma_t})\,A^T \vec{f}(t-1) & \text{if } t > 1 \land \exists\sigma_1 \end{cases} \tag{2.2}$$

Using more than one deviation detection mechanism

The diagnostic framework based on HMM presented in section 1.2.2 uses deviation detection information emitted by a single deviation detection mechanism; the same diagnostic framework can be exploited to deal with more than one deviation detection mechanism by applying the idea presented hereafter.

The idea is to slightly modify the way of using the set of symbols Σ and the associated observable symbol probability matrix B: instead of mapping each deviation detection message with a different symbol in Σ, we map each combination of deviation detection messages coming from different mechanisms with a different symbol.

Let's present the scenario in which two deviation detection mechanisms are used at the same time, X and Y. This scenario can be easily extended to encompass more than two deviation detection mechanisms. Let assume the following:

- X raises the following deviation detection messages: $\{x_1, \ldots, x_h\}$,
- Y raises the following deviation detection messages: $\{y_1, \ldots, y_k\}$.

For each pair of deviation detection messages (x_i, y_j) (for all $i = 1, \ldots, h$ $j = 1, \ldots, k$) we define the symbol $\sigma_{i,j} \in \Sigma$; if some pairs cannot hold because of intrinsic characteristics of the deviation detection mechanisms, no corresponding symbol need to be defined.

The new interpretation of set Σ slightly modifies the interpretation of matrix B, but the rationale and the formulas behind the diagnostic mechanisms are still valid. All the other extensions to the HMM diagnosis presented in this section are still applicable.

Dealing with fault probability values changing over time

One of the assumptions behind the fault model used to describe the diagnostic mechanism based on HMM presented in section 1.2.2 is that the probability values related to the occurrences of faults do not change over time. This assumption is well suited when dealing with the discrimination between transient and permanent hardware faults, but it is not well suited when the unpredictable behavior of a malicious attacker is considered.

Ideas to cope with this unpredictability go in the following directions:

1. Adapt over time the criteria used to take decisions after refreshing the state probability vector; this is a way to dynamically adapt the decision criteria, it is not a way to modify the evaluation criteria. This method requires the diagnostic mechanism to receive some (trusted) information from the system (e.g. from the reconfiguration sub–system) which can guide the adaptation of the decision criteria.

   For example, if the method of the threshold vector $\vec{d}$ is used (see section 1.2.2), some information coming from the reconfiguration sub–system (during a reconfiguration action it emerged that the triggering diagnostic alarm was a false alarm) can be used to adjust some threshold values within $\vec{d}$.

2. Apply the learning algorithm (see above) at some point in time, triggered by some trusted request which the diagnostic mechanism receives from the rest of the system. This method actually modifies the evaluation criteria used by the diagnostic mechanism when refreshing the state probability vector, but poses the problem of identifying which learning sequences have to be used for the setup of the internal parameter of the diagnostic mechanism.

3 CRITICAL INFRASTRUCTURE PROTECTION

This chapter describes the application of the principles described in chapter 2 to the protection of a critical infrastructure: the critical infrastructure used as a reference is the CRUTIAL[1] infrastructure, which is a SCADA–based infrastructure underlying the power production and distribution grid.

The architecture of the CRUTIAL infrastructure is presented [Veríssimo 08], where an implementation of the diagnosis framework proposed in section 2.2 was adopted. The CRUTIAL architecture encompasses the definition of information switches, the so–called CISs[2], where all the dependability–related activities are performed. The CIS resilience is achieved thanks to replication for intrusion tolerance and replica recovery for self–healing.

A quantitative analysis of the redundant architecture of the CIS is presented (part of this evaluation is in [Daidone 08]) with the objective of: i) identifying the relevant parameters of the architecture, ii) evaluating how effective is the trade–off between proactive and reactive recoveries, and iii) finding the best parameter setup.

The definition of the CIS bases on the assumption that the probability of intruding a CIS replica does not increase over time, which means that the attacker cannot acquire enough knowledge during its intrusive attempts in order to break the replica. This assumption can be supported by the extensive application of diversity during the recovery actions [Obelheiro 06], which is argument of the final part of this chapter.

The final part of the chapter presents the FOREVER[3] service [Bessani 08a], which is a service that can be used in systems replicated with diversity and rejuvenated by periodic replica recoveries in order to sustain a fundamental assumption: not–increasing probability of intrusion over time. The CRUTIAL architecture is a typical example of replicated architecture where the FOREVER service can be implemented.

The FOREVER service introduces the use of a set of operating systems' and applications' reconfiguration rules which can be used to modify the state of a system replica prior to deployment or in between recoveries, and hence increase the replicas chance of a longer intrusion–free operation.

1 CRUTIAL is a recent project funded by the IST programme of the European Commission (Contract IST-2004-27513). `http://crutial.erse-web.it/`
2 CRUTIAL Information Switches
3 FOREVER is a recent project funded by the EU through the RESIST NoE (Contract IST-2004-26764). `http://forever.di.fc.ul.pt/`

A quantitative analysis of the FOREVER service is presented, aiming to quantify how much the FOREVER service enhances the resilience of the system in which it is implemented; the analysis evaluates the probability of system failure through variation of i) time between recoveries, ii) penalty due when diversity is not applied, iii) probability of common vulnerabilities, and iv) mean effectiveness of configuration diversity rules applied.

3.1 THE PROTECTION OF AN ICT INFRASTRUCTURE CONTROLLING THE POWER GRID

Critical infrastructures as the power grid are basically physical processes controlled by computers interconnected by networks [Madani 05]. Some years ago those systems were highly isolated and hence secure against most security threats. During the last years the ICT part of those critical infrastructures evolved in several aspects: i) hardware and software devices (station computers, networks, protocols, ...) are no longer ad–hoc and proprietary, instead standard components (COTS) are used; ii) most of the station computers are connected to corporate networks and to the Internet.

These infrastructures are nowadays greatly exposed to cyber–attacks coming from the Internet (e.g. as documented in [Dawson 06] and [Wilson 06]), so they have a level of vulnerability similar to other systems connected to the Internet, but the socio–economic impact of their failure can be huge. This scenario, reinforced by several recent incidents (see the Introduction for a couple of examples), is generating a great concern about the security of these infrastructures, especially at government level (as shown in [Gordon 06]).

CRUTIAL project presentation

Recently the CRUTIAL project, funded by the IST programme of the European Commission, addressed the problem of the protection of the electric power grid; the project, which was active from the beginning of 2006 to the end of 2008, proposed new networked ICT systems for the management of the electric power grid, in which artifacts controlling the physical process of electricity generation and distribution need to be connected with information infrastructures, through corporate networks (intranets), which are in turn connected to the Internet.

Project solutions

The CRUTIAL project proposed an architecture encompassing trusted components in key places, which a priori induce prevention of some faults and of certain attack and vulnerability combinations; the remaining faults and intrusions are automatically tolerated by middleware devices supplying trusted services out of non-trustworthy components. Trustworthiness monitoring mechanisms are used for detecting situations not predicted and/or beyond assumptions made, and adaptation mechanisms are used to survive those situations. Finally the CRUTIAL architecture secures information flows with different criticality within/in/out of the critical infrastructure by using organization–level security policies and access control models.

3.2 CRUTIAL ARCHITECTURE OVERVIEW

This section describes in summary the CRUTIAL architecture [Veríssimo 08], which is a significant extension of previous intrusion–tolerant reference architectures (e.g. MAFTIA[4]) defined in order to deal with the specific challenges of the critical information infrastructure problem: use of legacy sub–systems, grant to global access control, requirement for non–stop operation and resilience.

The definition of the CRUTIAL architecture bases on the requirements of the power control system and on the assumptions about the malfunctions affecting the ICT part of the infrastructure which could "cascade" on the physical process controlled by the infrastructure itself.

Power controls

The power system is spread over a wide geographical area and requires some interconnection among a variety of information subsystem which are in charge of implementing quite sophisticated controls on the physical process; power controls are typically arranged in the following hierarchical structure:

PRIMARY CONTROL: it concerns the local control of generators, implemented by special-purpose electronics and programmable controllers. The primary control could be performed without the need of communication with the rest of the power system, just seeking to maintain equilibrium on local electrical and mechanical parameters.

SECONDARY CONTROL: it aims to regulate the power production/distribution inside a given area, according to some strategy; each strategy dynamically sets some local parameters about power generation to be commanded to the primary control. The secondary control needs relatively fast communications among area's substations, and it is vital for the power system.

TERTIARY CONTROL: it aims to optimize energy losses and marketing aspects, so it does need communications. The tertiary control is not vital for the power system management; in fact, if it were inactive, the power grid would be still operative, albeit in a sub–optimal condition.

ICT malfunctions

The ICT part of the power infrastructure is affected by malfunctions classified as follows:

HARDWARE FAULTS: they occur in different physical components leading to the following problems:

4 Malicious–and Accidental–Fault Tolerance for Internet Applications

- Problems as *undue tripping*[5] or *missing action* in the devices directly related to the primary control of the power system (SCADA devices);
- At communication level, leading to lost packets (packet lost on a dying link) or late packets (some packets could be routed on too long paths);
- In the computers performing the secondary (and the tertiary) control or supporting corporate networks; those faults can lead to the crash of a part of the information system or to erroneous command sequences (value faults).

COMMUNICATION RELATED FAULTS: omitted messages, burst losses (due to intermittent physical disconnections), late messages, unexpected messages, wrong values, Byzantine faults, network partitioning.

SOFTWARE RELATED FAULTS IN THE INFORMATION NODES: most faults at this level can manifest as application/system crashes, but the possibility of a less favorable behavior is not ruled out (e.g. issuing wrong and/or inconsistent messages just before crashing). Examples of software faults are application errors, O.S. related errors, resources exhaustion, late services. . .

MALICIOUS FAULTS: these malfunctions can be further classified as follows:

- *Attack*: malicious interaction fault through which an attacker aims to deliberately violate one or more security properties (an attack is an intrusion attempt); an example of malicious attack is the DoS.
- *Vulnerability*: a security hole left out during the development of the system or opened during operation.
- *Intrusion*: a malicious, externally-induced fault resulting from an attack that has been successful in exploiting a vulnerability; examples of intrusions are Trojan horses, worms, viruses. . .

A security failure at one level of decomposition of the system may be interpreted as an intrusion at the next upper level (e.g. the failure of an authentication and authorization mechanism to prevent system penetration by a malicious user is an intrusion as seen from the containing system). The concept of "Intrusion tolerance" can be introduced (how to provide correct service in the presence of intrusions): Intrusion Containment Regions (ICRs) could be introduced by analogy with the notion of Fault Containment Regions (FCRs) in order to guarantee certain security properties despite of the fact that some components could be compromised.

WAN–of–LANs The definition of the overall CRUTIAL architecture bases on the observation

5 Tripping is the action of opening a breaker; this action is often a local protection action used to isolate a faulty grid segment from the rest of the grid.

that (i) all the ICT devices necessary for the control of the whole power grid are logically grouped in substations and finally LANs[6], and (ii) LANs are interconnected by a global interconnection network, called WAN[7], which is a logical entity owned and operated by the operator companies of the critical information infrastructure. Those companies may or not use parts of public networks (e.g. Internet) as physical support, so relevant is the problem of DoS attacks[8], which would negatively affect the remote control of industrial applications (e.g. the secondary and tertiary control of the power grid).

Since all the traffic originates from and goes to a LAN, the CRUTIAL architecture represents facilities as protected LANs interconnected by a WAN (see figure 3.1); using such an architecture, the problem of protecting the power grid (and similar critical infrastructures) is reduced to the problem of protecting LANs from the WAN or other LANs.

CRUTIAL Information Switch (CIS)

Each LAN is connected to the WAN through a special interconnection and filtering device, the CRUTIAL Information Switch (CIS), which ensures that both the incoming and outgoing traffic satisfies the security policy[9] defined to protect the infrastructure. The rationale behind the commitment of the protection to the CIS is the requirement of minimizing the updates of/modifications to the existing legacy machinery. Using the CIS, the only update required in a station computer is the use the IPsec[10] protocol (instead of IP[11]) in order to force the station computer to accept only traffic forwarded by its CIS.

A CIS is a kind of improved firewall that works at the application layer and that is required to be intrusion tolerant in order to guarantee continual operation to the power grid.

The WAN–of–LANs architecture allows the definition of areas with different levels of trustworthiness: considering how it is easy to define a LAN[12] in today' IP architectures (e.g. by using Virtual switched LANs), the rationale behind the WAN–of–LANs can be hierarchically re-iterated, so there is virtually no restriction to the level of granularity for the definition of the protected areas. In consequence, the CRUTIAL architecture allows to deal with both outsider threats (protecting a facility from the Internet) and insider threats (protecting a critical host from other hosts in the same facility by locating them in different LANs).

6 Some examples are the administrative clients and the servers LANs, the operational (SCADA) clients and servers LANs, the engineering clients and servers LANs, the Public Switched Telephone Network (PSTN) modem access LANs, the Internet and extranet access LANs, etc.

7 Wide Area Network

8 DoS attacks are nowadays one of the most serious security threats to the Internet.

9 A security policy is defined by the Common Criteria as the set of laws, rules, and practices that regulate how an organization manages, protects, and distributes sensitive information. `http://www.commoncriteriaportal.org/`

10 Internet Protocol Security

11 Internet Protocol

12 Local Area Network

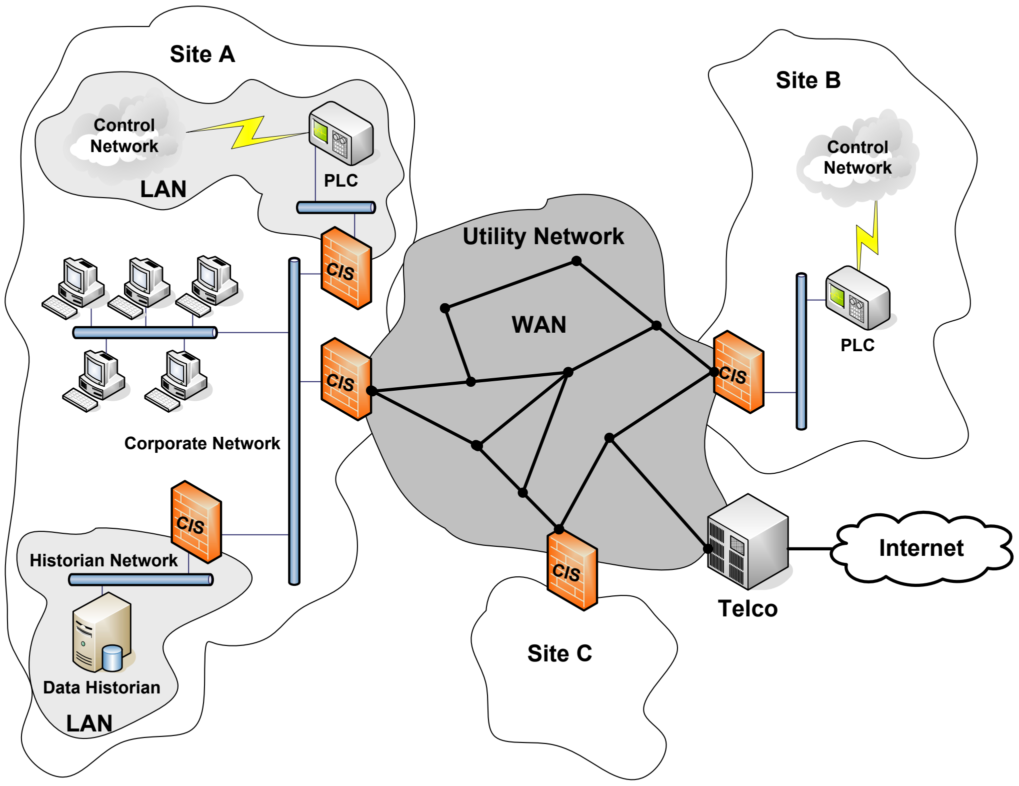


Figure 3.1: The CRUTIAL architecture: a WAN of LANs, where each LAN (e.g. an entire site) is connected to the WAN through a special device called CIS.

The distributed system made up by CISs

CISs collectively act as a set of servers providing distributed services aimed to control both the command and information flow among the ICT parts of the critical infrastructure, securing a set of necessary system–level properties. This set of cooperating servers must be intrusion–tolerant, prevent resource exhaustion providing perpetual operation, and be resilient against assumption coverage uncertainty, providing graceful degradation or survivability.

CIS architecture

The CIS architecture closely follows the node structuring principles for intrusion–tolerant systems presented in [Veríssimo 06b]; the CIS architecture, shown in figure 3.2, is composed by the architectural (macro–) levels described in the following.

The *Hardware* level encompasses the node and networking devices that make up the physical distributed system; hardware is divided into two parts, a *trusted* and an *untrusted* one. The assumption is that most of the node's operations run on untrusted hardware (e.g. the usual machinery of a personal computer), connected through the normal networking infrastructure (which is called *payload channel*). The trusted part is formed by an appliance board with a processor and (possibly) a network adapter connected to a control channel; this part is trusted, for example, because intruders do not have direct access to it by construction.

The *Local Support* level encompasses the following components: a *Trusted Software* component, which executes a few critical functions correctly (the rest being subjected to malicious faults), the *Operating System*, and a *Run–Time Environment*, which offers both trusted and untrusted software and operating system services in a homogeneous way. This level encompasses the *Proactive–Reactive Recovery Wormhole* (PRRW) service, which manages the periodic and event triggered recovery actions on the CIS replicas. The PRRW service will be detailed in the following sections.

The third macro–level is *Distributed Software*, where the distributed software provided by CRUTIAL is running; the distributed software is divided into a middleware layer on top of which distributed applications run, even in the presence of malicious faults.

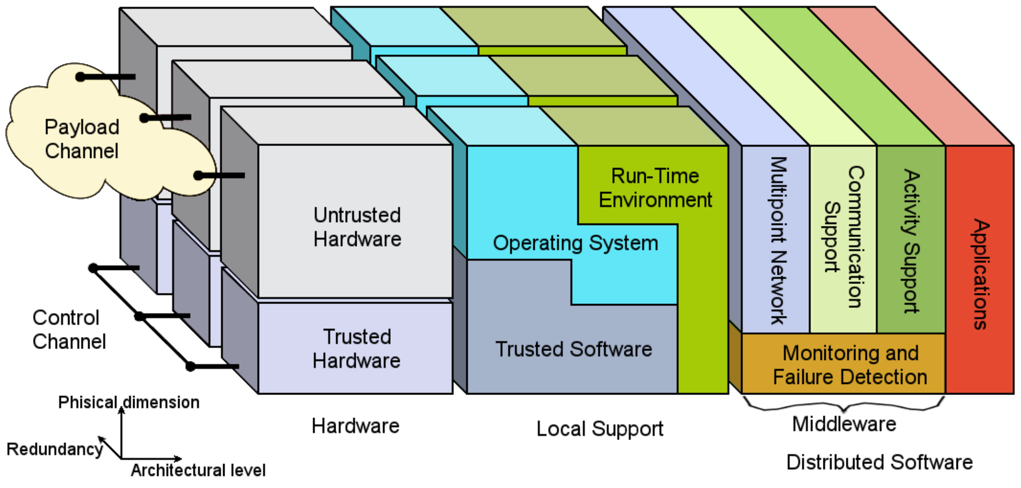


Figure 3.2: The CIS architecture.

Middleware services

The intrusion–tolerant middleware encompasses four modules (see the right–most part of figure 3.2), each one providing services to other modules or directly to the applications:

MULTIPOINT NETWORK: this is the lowest module within the CRUTIAL's middleware, and features an abstraction of basic communication services (e.g. IPsec, TCP, SSL/TLS).

COMMUNICATION SUPPORT: this module comprises basic cryptographic primitives, Byzantine agreement, consensus, group communication and other core services; in particular the following services are provided.

The *Randomized Intrusion-Tolerant Services* (RITAS), which are organized as a stack of randomized intrusion–tolerant protocols, support applications which depend on intrusion–tolerant broadcast and agreement; these protocols, being randomized, overcome the impossibility result in asynchronous settings established in [Fischer 85] (also called the FLP result), but present a significant performance improvement over previous protocols of the same class.

The *Communication Service* supports secure communication between CIS and, ultimately, between LANs; it provides secure channels, multicast primitives, and probabilistic gossip–based information diffusion between CIS.

The *Fosel Service* mitigates DoS attacks by using an overlay protection layer on top of the normal infrastructure.

ACTIVITY SUPPORT: this module comprises the following services:

The *Protection Service* protects areas from one another, i.e. a LAN from another LAN or from the WAN, thus allowing the treatment of both outsider and insider threats; in particular the Protection Service protects the station computers by filtering the messages directed to them.

The *Access Control and Authorization Service* defines the rules for collaboration and information exchange between sub–modules of the architecture, corresponding in fact to different facilities of the critical infrastructure; this service defines the security policy[13] for the infrastructure.

MONITORING AND FAILURE DETECTION: this module implements functionalities related to monitoring and failure detection. This module assesses the connectivity and correctness of remote nodes, and the liveness of local processes; trustworthiness monitoring and dependable adaptation mechanisms also reside in this module, and have interactions with all the middleware modules (both the Activity Support and Communication Service modules depend on those mechanisms).

3.2.1 *CIS Resilience Overview*

CIS resilience is achieved thanks to replication for intrusion tolerance and replica recovery for self–healing [Sousa 06, Sousa 07]: replication is used in order to guarantee system correct operation when some replicas are compromised, rejuvenation is instead used primarily to remove the effects of malicious attacks aiming to compromise some replicas and to break the system.

Replication with diversity

The CIS is replicated (with diversity) in n machines and follows its specification as long as at most f of these machines are attacked and behave maliciously, both toward other replicas and toward the station computers in the protected LAN. Given that the replicated CIS accepts (and forwards) a message if the message is accepted by at least $f+1$ replicas, at least $n \geqslant 2f+1$ replicas are required. Replicas are diverse in order to substantiate the following, fundamental

13 The security policy is managed by using PolyOrBAC, the web–services–based version of OrBAC (Organization Based Access Control) presented in [Abou El Kalam 07].

assumption: fault independence for the replicas. Each replica uses a different operating system (e.g. Linux, FreeBSD, Windows XP), and all the operating systems are configured to use different passwords and different internal firewalls (e.g. IPtables, IPF, Windows firewall). More discussion about how to support diversity among replicas is given in section 3.5.

Each replica is connected to the LAN and to the WAN through two *Traffic Replication Devices* (as shown in figure 3.3) which behave like Ethernet hubs: when they receive a packet from a port, they broadcast it to all the other ports. Looking at the traffic directed toward the station computers in the LAN, the WAN side traffic replication device spreads the packets received from the WAN to all the replicas, whilst the LAN side traffic replication devices spreads the traffic generated by each replica to all the other replicas and to the station computer in the LAN. There is a distinguished payload replica, the so–called *leader replica*, which is in charge of forwarding messages to the station computers: this avoids unnecessary traffic multiplication on LAN side (e.g. *n* copies of the same message forwarder to the destination node).

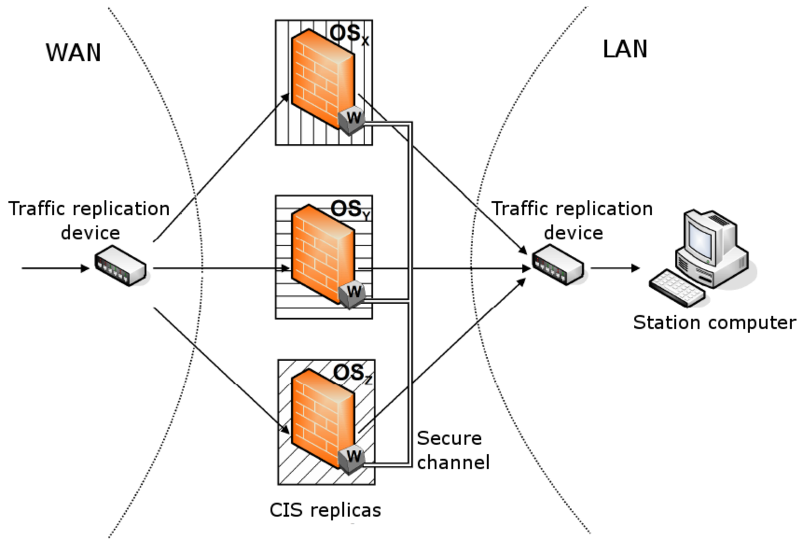


Figure 3.3: The hybrid and replicated (with diversity) architecture of the CIS.

Replica recoveries

CIS intrusion tolerance is enhanced by rejuvenating CIS replicas through recoveries, as presented in [Sousa 07]; the replica rejuvenation strategy, called PRRW[14], is based both on periodic (proactive) recoveries and on event triggered (reactive) recoveries, seeking perpetual unattended correct operation. The key property of the PRRW strategy is that, as long as the fault exhibited by the replica is detectable, this replica will be recovered as soon as possible, ensuring that there is always an amount of replicas available to sustain correct operation [Sousa 07]. In order to guarantee system availability despite the unavailability

14 Proactive–Reactive Recovery Wormhole

of recovering replicas, the number of replicas has to be $n \geqslant 2f + 1 + k$, where k is the maximum number of replicas allowed to recover in parallel; this way the system is able to tolerate at most f Byzantine replicas and recover k replicas simultaneously.

Recoveries have beneficial effects (e.g. reactive recoveries rejuvenate replicas detected as incorrect), but also negative effects (e.g. the proactive recovery of a correct replica makes the replica unavailable for the whole duration of the recovery); this issues are some of the arguments of the analysis about the resilience of the CIS redundant architecture presented in section 3.4.

Hybrid architecture

The CIS is implemented using an hybrid architecture, so it is composed by two parts: the *payload* and the *wormhole* [Veríssimo 06a]. The payload is an asynchronous system where applications and protocols are executed; the wormhole is a secure and synchronous system providing services to the payload part through a well-defined interface (e.g. it triggers replica recoveries, it executes a simple voting protocol). The wormhole part of each replica, called *local wormhole*, is connected to the other local wormholes through a synchronous and secure control channel, isolated from other networks.

3.3 DIAGNOSIS IN CRUTIAL

This section describes how the diagnosis activity was designed in CRUTIAL.

The CRUTIAL infrastructure is organized as a WAN–of–LANs, where each LAN is connected to the WAN by adding a new machinery to the infrastructure: the CIS; given that the computers inside the LANs cannot be modified/updated, all the diagnosis activity related to the infrastructure has to be performed inside the CIS.

Diagnosis scenarios

Diagnosis in CRUTIAL is implemented at middleware level and follows the diagnosis framework presented in section 2.2, so that the following diagnosis scenarios arise [Veríssimo 08]:

CIS SELF-DIAGNOSIS (local view): this is the diagnosis activity performed locally to each CIS, aiming to monitor the CIS itself (e.g. to diagnose hardware faults, intrusions. . .).

LAN DIAGNOSIS (private view): this is the diagnosis activity performed locally to each CIS, aiming to monitor the nodes that are inside the LAN and that are protected by the CIS itself (e.g. to "measure" the trustworthiness level of a certain node).

CIS DISTRIBUTED DIAGNOSIS (global view): the CISs in the WAN construct a common view about the state of a certain CIS in the infrastructure (e.g. the liveness and trustworthiness of a specific CIS).

Monitoring and failure detection strategy

From a system–level viewpoint, the monitoring and failure detection strategy is organized as follows:

- Every CIS diagnoses itself over time, in order to judge the healthy/faulty status of its resources/services, primarily for local reconfiguration aims; this step is the result of the activity of a CIS Self-Diagnosis service implemented in the middleware.
- Every CIS (when asked for) declares its "health status" (full report, only relevant parts or a signature of them). Since CISs are not completely trusted by the other CISs, they do not completely trust the declared "health status", so they also try to build a private perception of the other CISs, basing on the possible direct relationships with them. The "declared status" and the "perceived statuses" could be conflicting (e.g. because of communication problems or because of deliberate and malicious causes).

  When CIS_A needs to use some remote resources/services on CIS_C, but CIS_A has no private perception of CIS_C, gossip can be applied: if CIS_A trusts CIS_B and CIS_B has a private perception of CIS_C, the private perception of CIS_C as seen by CIS_A can inherit the private perception of CIS_C as seen by CIS_B.
- When necessary, e.g. when the private perception is not enough for some reason, (pertinent groups of) CISs exchange among them their own private perceptions of a certain resource, in order to reach an agreement about that. The result of the agreement overrides the result of the private diagnosis.

3.3.1 *CIS self-diagnosis (local view)*

From a local viewpoint the CIS is a sophisticated application level firewall, combined with equally sophisticated intrusion detectors; the CIS is hence required to be intrusion tolerant, to prevent resource exhaustion providing perpetual operation and to be resilient against fault assumption coverage uncertainty providing survivability. In order to comply with all the above requirements, the CIS has a hybrid architecture and is replicated (with diversity) in *n* replicas (more details in [Bessani 07]). Each CIS replica is built using a synchronous and secure local wormhole and an asynchronous and insecure payload.

Two monitoring/failure detection scenarios arise:

INTERNAL MONITORING: monitoring performed inside a single replica, trying to detect local failures (e.g. an intrusion).

EXTERNAL MONITORING: monitoring performed on the perceived behavior of the other replicas (e.g. to detect a replica crash).

Internal monitoring

The internal monitoring, given the information system malfunctions introduced in section 3.2, has to be done on the following components/services:

- Hardware components (e.g. network interfaces, processing units, memory modules. . .) which are supporting the replica. The monitoring activity on these components makes sense only when physical replication is used; in case of logical replication, these components need to be monitored in the host system running the replicas and hence outside the CRUTIAL middleware.

- Software components belonging to several architectural levels in the payload or in the operating system.

Several signals coming from many architectural levels are collected and processed over time: an example of signal coming from low architectural levels (O.S.) is related to a CPU fan that is working too slow or a temperature sensor that is signaling the CPU is too warm. An example of signal coming from a higher architectural level is an application–generated exceptions or error return code.

The internal monitoring activity has hence to identify compound system conditions which could require diverse corrective actions; for example, repeated application errors could be interpreted as manifestation of software aging requiring rejuvenation, or could be correlated with lower level signals (the CPU is too warm because the CPU fan is working too slow), requiring another kind of reconfiguration (e.g. replacing the CPU fan). The rationale behind internal monitoring and failure detection is to try to stop and repair the faulty replica before it starts to behave incorrectly.

External monitoring The external monitoring is performed by each replica on the perceived behavior of the other replicas, given that a replica is not guaranteed to always behave correctly. The monitoring activity is performed at service level, so that each service is in charge of detecting whether its peers running in the other replicas seems correct or not. An example of middleware service monitoring its peers on other replicas is the CIS Protection Service.

How to use SEC for CIS Internal Self-diagnosis

CIS internal self-diagnosis was implemented using SEC (see section 1.3.2) and the conceptual schema described in section 2.3; this subsection reports considerations about this implementation.

How to deal with diversity The CIS is replicated with diversity, including a different operating systems for each replica: SEC is written in PERL, so it can be executed on several operating system platforms by using the specific PERL interpreter. SEC requires text–formatted streams of detection information as input: this is feasible, especially for system logs, which are typically text based. For example, Linux based operating systems support the "syslog" log file; for other non–text systems log files, free tools are available to support the same logging system[15]. In order to speed–up the recognition of urgent signals, a special stream can be created ad–hoc (let's call it *emergency log*) and the sensor driver can be forced to generate a log event not only in the log of the operating system, but also in the emergency log.

SEC performs the following activities:

- It processes streams of information about events occurring in the system by reading log files on the fly (as the *Collector* is supposed to do);
- It triggers actions when specific events are recognized based on some rules (as the *Aggregator* is supposed to do);
- It correlates events on the base of some (originally static) rules. Rules are defined in specific configuration files (text format) called *rule files*; rules can be based on several properties of events, e.g.:

  a relative timing;

  b localization: each physical event generator is cataloged in a parallel-hierarchic data structure, reflecting the physical position - e.g. a CPU is located on a circuit board, which is in a card cage, which is in a server rack, which is in a room, etc. The distance between two elements is defined by traversing the structure along a path joining these elements.

  c architectural levels: a structure similar to b) above, where the neighborhood is in general expressed in terms of interactions: two components directly interacting are neighbor, if their interaction is mediated through 3 levels of other components their distance is proportional to 3, and so on.

  SEC contexts are used to implement the recognition of the poly–events introduced in section 2.3.

How to self–learn

SEC is self–learning, because configuration files can be refreshed at run–time, keeping the status of the ongoing correlations. SEC can in fact be restarted from the hosting operating system using specific inter–process signals (e.g. `SIGABRT` or `SIGTERM`), possibly performing a "soft" reset, so that configuration files (rule files) are reloaded, and the status of the ongoing correlations is restored. In particular, a SEC soft restart consists of the following steps:

1. Rule files are reopened (new files can be opened too);
2. Event correlation operations started from rule files that have been modified or removed after the previous configuration load are canceled;

15 For example, the freely available "Snare for Windows" tool (from Intersect Alliance) can be used to convert Microsoft event logs to "syslog" messages.

3. Other operations and other event correlation entities (contexts, variables, child processes, etc.) remain intact.

External monitoring: diagnosis through the Protection Service

The Protection Service (PS) introduced in section 3.2 is the middleware service performing egress/ingress access control in the CIS, implementing an instance of the global security policy. The PS works on the traffic crossing the CIS in both directions: from the WAN to the protected LAN and from the protected LAN to the WAN (and finally to a remote LAN); the focus of this work is in the first direction, the WAN–to–LAN one, which is the most critical out of the two.

Protection Service The PS verifies whether the messages received by the CIS from the WAN side comply with the security policy, forwarding those satisfying the security policy to their destination node in the protected LAN; messages not satisfying the security policy are discarded. The PS instance running in a CIS replica is executed in the payload; as soon as it receives an incoming message, it notifies its (positive) approval to its local wormhole. The wormhole collects message approvals coming from the local wormholes and decides whether each message is valid or not: an incoming message m is considered *valid* if and only if the wormhole collects at least $f+1$ different (positive) approvals for m. Valid messages are forwarded to their destination by a distinguished payload replica only, the so–called *leader replica*, in order to avoid unnecessary traffic multiplication on LAN side. The management of the leader replica is performed by the PRRW service with the support of the wormhole.

Monitoring Monitoring is performed at payload level, where each instance of the PS checks whether other replicas behave correctly, triggering specific accusations when necessary. Each payload replica has in fact to verify whether all the signed messages are forwarded to the protected LAN by the leader replica, and to check whether invalid messages are sent toward the protected LAN. Each (correct) replica i expresses accusations about replica j by using the following function calls of the local wormhole:

- `W_detect(`j`)`: replica i detects that replica j is faulty; this is the case in which replica i receives a message m sent by replica j and verifies whether m is illegal.
- `W_suspect(`j`)`: replica i suspects that replica j is faulty; this is the case in which replica i verifies whether replica j, being the leader, is not forwarding a legal message to the protected LAN.

Diagnosis Fault diagnosis is performed at wormhole level, where accusations raised by the replicas are collected and interpreted on the basis of the following quorums:

- Replica j is diagnosed to be maliciously faulty as soon as $f+1$ detections about j are collected (that is at least $f+1$ local wormholes received a

`W_detect(`j`)` function call); in this case, at least one correct replica detected replica j to be maliciously faulty.

- Replica j is suspected of being faulty as soon as $f+1$ accusations (detections and/or suspects) are collected; in this case, at least one correct replica raised a suspect about replica j being faulty.

Reconfiguration

The reconfiguration strategy is executed by the PRRW service triggering an *immediate* reactive recovery in case of a replica diagnosed maliciously faulty and planning a *delayed* reactive recovery in case of suspect.

3.3.2 *CIS LAN diagnosis (private view)*

The CIS monitors over time the nodes in its protected LANs in order to evaluate their trustworthiness; the evaluated trustworthiness level is then used to apply the proper reconfiguration action on the protected untrusted nodes (e.g. for replacing hardware, refreshing the software, changing passwords, ...).

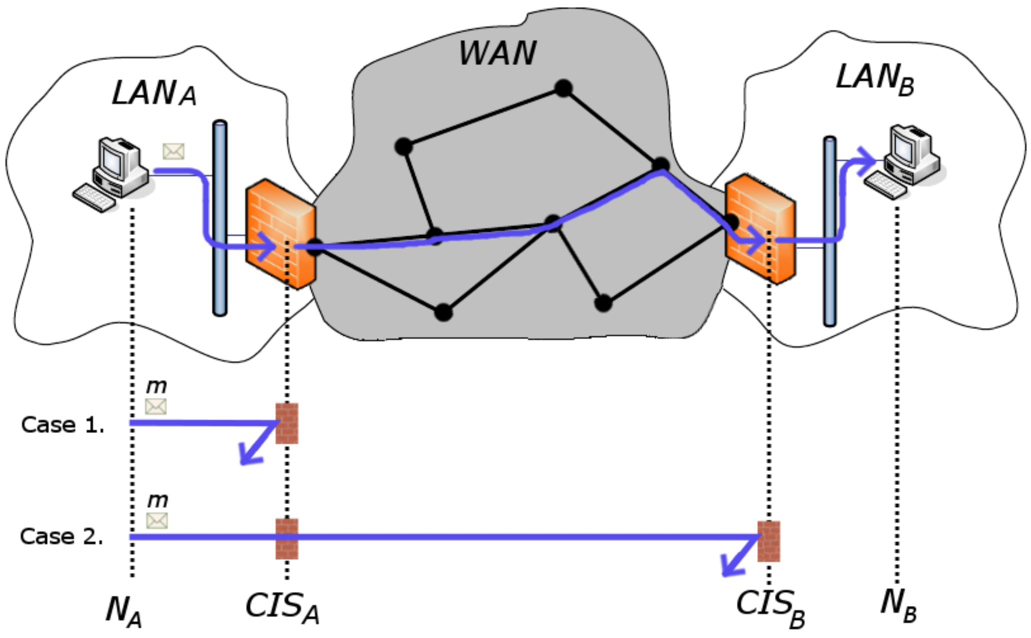


Figure 3.4: Detection scenarios for LAN diagnosis.

The trustworthiness indicator

A trustworthiness indicator for each node N_A protected by CIS_A is defined (it could be could be multi–dimensional). The trustworthiness indicator is modified based on the following detections:

1. The instance of the security policy applied within CIS_A to the traffic originated from the protected LAN detects that message m, sent by node N_A, violates the security policy (e.g. node N_A is trying to send a command to a

remote station computer node N_B without being allowed to do it). In this scenario, depicted in figure 3.4 as "Case 1.", the trustworthiness indicator for N_A is updated accordingly and CIS_A drops message m.

2. The instance of the security policy running on the remote CIS_B detects that message m, sent by node N_A to node N_B, violates the security policy. CIS_B notifies CIS_A this violation and drops message m (this scenario is depicted in figure 3.4 as "Case 2.").

   CIS_B discriminates whether the incoming message m really comes from a station computer (instead from an hacker in the WAN) by using the LAN Traffic Labeling service (the message must be signed by CIS_A, which protects the source node N_A). The signed label is hence a proof of the source of the message.

The LAN diagnosis service collects over time the above detections in order to evaluate the trustworthiness indicator of each protected node. If the trustworthiness indicator of a protected node N_A exceeds a given threshold, the LAN diagnosis service alerts its peers about node N_A being untrusted, so that adequate countermeasures can be taken.

3.3.3 *CIS distributed diagnosis (distributed view)*

The several replicas that made up a single CIS are required to perform the same operations; this simplifies somewhat the task of checking their correctness on the run. Each single CIS, as seen from the WAN, is a different logical entity, in terms of actions, services and requests toward other CISs. In the ordinary information flux there is no simple comparison rule check that can be performed, to catch on the fly a mischievous partner. On the other hand, if a CIS becomes compromised, internal redundancy and resilient architecture notwithstanding, then necessarily the basic hypothesis on the fault occurrence has been broken: more than f replicas are out of order together. Of course, this is the catastrophic case, whose probability has to be lowered down to a target level by choosing proper redundancy figures. However, a local catastrophe (regarding a single LAN controlled by a compromised CIS) not necessarily should imply the downing of the entire system. In fact, on the WAN side, all CISs attempt to maintain a common view of two parametric descriptors of the health of their partners: *liveness* and *trustworthiness*.

LIVENESS: it is checked in two ways: i) passively, by monitoring normal network traffic from the target; ii) actively (when the former is not frequent enough), by exerting a form of resilient ping, by means of a simple challenge/response protocol.

TRUSTWORTHINESS: it is built up by checking the formal correctness of the messages coming from the target, as well as from any access violation detected by the CIS Protection Service.

3.4 QUANTITATIVE EVALUATION OF THE CIS RECOVERY STRATEGY

This section presents the quantitative analysis of the redundant architecture of the CIS (part of this analysis is in [Daidone 08]); the objectives of the analysis were the following: i) evaluate how effective is the trade–off between proactive and reactive recoveries, ii) identify the relevant parameters of the architecture, and iii) find the best parameter setup.

Two dependability and availability measures of interest were identified; a model of the recovery strategy was constructed in order to analyze the quantitative behavior of the recovery strategy. The impact of the detection coverage, of the intrusions and of the number of CIS replicas on the measures of interest was analyzed and discussed, aiming to evaluate how effective is the trade–off between proactive and reactive recoveries.

The analysis showed that the increment of the detection coverage of intrusions has conflicting effects on both dependability and availability measures, and that these effects depend also on the behavior of invalid or omissive intrusions. The analysis results suggested some directions for refining and improving the recovery strategy.

3.4.1 *Fault Model and Assumptions*

This section describes the fault model and the assumptions on which the fault model is based on; all this information was extracted from the description of the recovery strategy given in [Sousa 07].

Station computers are assumed to only accept messages signed by the wormhole (a symmetric key K is shared between the station computer(s) and the CIS wormhole).

Faults

The following faults are considered:

f1) The faults related to communication involve both the traffic replication devices and the communication channels among them and the replicas (except the control channel connecting local wormholes). Traffic replication devices can lose messages coming from a port or delay the traffic forwarding on some ports (for an unbounded time); traffic replication devices cannot generate spurious messages or alter messages. Communication channels can lose messages or unpredictably delay the traffic forwarding.

f2) A payload replica can be intruded, and hence can be affected by Byzantine faults.

f3) A local wormhole can only fail by crash; at most $f_c \leqslant f$ local wormholes are assumed to fail by crash. The crash of a local wormhole is detected by a perfect failure detector. When a local wormhole crashes, the corresponding payload is forced to crash together.

f4) Fault-independence is assumed for payload replicas, i.e. the probability of a replica being faulty is independent of the occurrence of faults in other replicas; this assumption can be substantiated in practice through the extensive use of several kinds of diversity [Obelheiro 06], as presented in section 3.5.

f5) The same attack on the same replica has always the same probability of success; also this assumption can be substantiated in practice through the extensive use of several kinds of diversity [Obelheiro 06], as presented in section 3.5.

f6) Station computers cannot be compromised (it is the trusted network that we aim to protect, exactly in the sense of preventing it from being compromised).

f7) Replicas are correct after their recovery.

f8) The security policy verified by the CIS is assumed to be perfect; this means that a correct replica applies perfectly the policy verification and there are no policy inconsistencies between replicas (i.e. all correct replicas verify the same policy).

Failure modes Given the faults described above, the corresponding failure modes for a payload replica are the following:

CRASH: the payload replica crashes because of the crash of the corresponding local wormhole (f3) or as the effect of an intrusion (f2).

OMISSION: the payload replica is subjected to a transient omission because of communication problems (f1) or as the effect of an intrusion (f2). For example, a transient omission occurs when the leader payload is not forwarding a signed message because it never received it from the traffic replication device (f1).

INVALID: the payload replica is failing by value as the effect of an intrusion (f2), e.g., it is sending illegal messages toward the LAN or it is flooding the WAN and the LAN aiming to delay the forwarding of legal messages.

For ease of modeling, we assume that a replica, as soon as it is successfully intruded, explicitly manifest failures (of any kind); this assumption is justified by the fact that until the replica behaves correctly (and hence it is not failing) despite of the undetected intrusion, the overall system behaves correctly. We assume also that a failure caused by an intrusion is permanent.

System unavailability

The system is unavailable if the number of correct working replicas is less than $f+1$ (so quorums cannot be reached) or if there are more than $f+1$ correct replicas, but the leader is omitting (so legal messages are not forwarded).

System failure

The system fails if the number of invalid replicas exceeds f (the correctness of the system cannot be guaranteed) or if the necessary resources are unavailable for a fixed duration[16] (CIS seeks perpetual operation).

3.4.2 *The PRRW Strategy*

This section describes the PRRW[17] strategy as originally defined in [Sousa 07]. The quantitative analysis presented in the following sections analyzes mainly the PRRW strategy, but also some variants derived from PRRW; the description of the PRRW variants considered will be given in section 3.4.3, where the evaluation results are presented.

Strategy parameters

The PRRW strategy manages the CIS replica recoveries using a mix of proactive and reactive recoveries, and it is characterized by the following parameters:

- T_P, the maximum time interval (cycle or recovery period) between consecutive recoveries on the same replica (each replica is hence recovered at most after T_P).
- T_D, the (worst case) execution time of a recovery.
- k, the maximum number of replicas that may recover simultaneously.
- f, the maximum number of simultaneously corrupted replicas that the system can tolerate.

Recovery scheduling

The PRRW strategy is scheduled as shown in figure 3.5: time is divided in $\lceil n/k \rceil$ different time slots that are cyclically repeated. Each slot is divided as follows: sub–slots S_{ij}, grouped under the unique name A, and sub–slot R_i ($i = 1, \ldots, \lceil n/k \rceil, j = 1, \ldots, \lceil f/k \rceil$).

Proactive recoveries

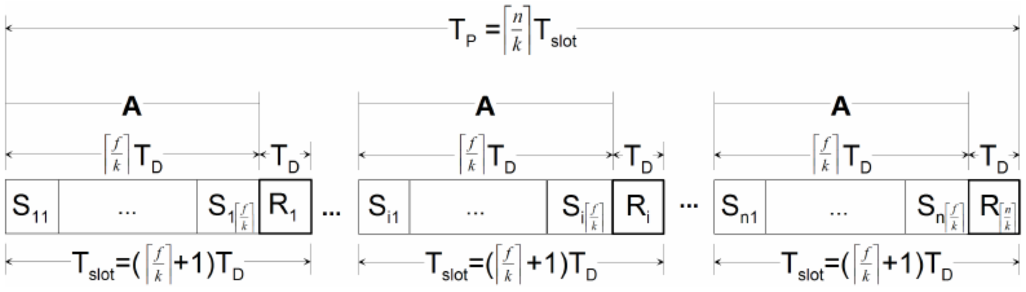
Proactive (periodic) recoveries are executed during sub–slot R_i only; up to k replicas recover simultaneously in each sub–slot R_i, according to the replica index. Replica i, with $i = 1, \ldots, k$, are recovered in sub–slot R_1, replica i, with $i = k+1, \ldots, 2k$, are recovered in sub–slot R_2 and so on. Sub–slot R_i lasts for (at most) T_D and it is executed again after a period T_P.

Reactive recoveries

Two types of reactive (a-periodic) recoveries can be triggered on replica i:

16 Let T_O be the duration of the mentioned time interval.

17 Proactive–Reactive Recovery Wormhole



$T_P = \lceil \frac{n}{k} \rceil T_{slot}$
A
$\lceil \frac{f}{k} \rceil T_D$
T_D
S_{11}
...
$S_{1\lceil \frac{f}{k} \rceil}$
R_1
S_{i1}
$S_{i\lceil \frac{f}{k} \rceil}$
R_i
S_{n1}
$S_{n\lceil \frac{f}{k} \rceil}$
$R_{\lceil \frac{n}{k} \rceil}$
$T_{slot}=(\lceil \frac{f}{k} \rceil+1)T_D$

Figure 3.5: The scheduling of recoveries in the PRRW strategy; proactive recoveries are executed during the R_i sub-slots, reactive *delayed* recoveries are executed during the S_{ij} sub-slots.

1. *Immediate* reactive recovery, triggered if a quorum of $f + 1$ accusations exists about i sending illegal messages; in this scenario replica i is *detected* of being compromised, because at least one correct replica detected that replica i is failed.

2. *Delayed* reactive recovery, triggered if a quorum of at least $f + 1$ accusations exists about the current leader i, some about i sending illegal messages, other about i not forwarding a signed message (the signed messages was not forwarded for more then O_t times). In this scenario the leader replica i is *suspected* of being compromised, because at least one correct replica raised an accusation about leader replica i, but the wormhole is not able to identify which accuser replica is correct, so it is not able to identify which kind of accusation is correct about leader replica i.

Immediate reactive recoveries are immediately triggered on replica i as soon as the replica is detected of being compromised.

Delayed reactive recoveries are triggered on the leader replica only, are executed during the sub–slots belonging to group A and are coordinated with proactive recoveries. If no immediate reactive recovery is already triggered for replica i, the PRRW strategy finds the closest recovery sub-slot where the recovery of replica i does not endanger the availability of the CIS. If the found sub-slot is located in the slot where replica i will be proactively recovered, the delayed reactive recovery is not performed. Each group A is divided into $\lceil f/k \rceil$ recovery sub–slots, identified as S_{ij}, and up to k replicas can be recovered simultaneously in each of these sub–slots. Group A lasts for (at most) $\lceil f/k \rceil T_D$.

Each slot lasts hence for up to $(\lceil f/k \rceil + 1)T_D$ with period T_P. After each R_i sub–slot has been executed once, each replica has been proactively recovered once.

A new leader is elected by the wormhole if the current leader is recovering or if the local wormhole of the current leader is detected to be crashed. The new leader is chosen as the (currently not crashed) replica more recently recovered by a proactive recovery.

3.4.3 *Quantitative Analysis*

This section presents a quantitative analysis of the PRRW strategy and of some PRRW variants. The relevant measures of interest are identified and the relevant parameters are described; the model representing the PRRW strategy is described and finally the results of the performed simulations are presented and discussed.

Proactive vs. reactive recoveries

The quantitative analysis of the PRRW strategy aims to evaluate how effective is the trade–off between proactive and reactive recoveries. Proactive recoveries rejuvenate the replicas in predefined instants of time, without being based on any fault detection. This means that proactive recoveries treat all the faults, including also the latent and hidden ones, which cannot be treated in other way, but they recover also correct replicas, weakening the availability of the system. On the other side, reactive recoveries are triggered only on replicas detected or suspected of being faulty; replicas not detected or suspected of being faulty are never recovered, even if they are actually faulty, weakening the dependability of the system.

A multiple phased system

Recoveries determine a discontinuity in the CIS configuration caused by the temporary unavailability of the replicas subjected to a recovery. Therefore it is possible to represent the entire operational life split into different periods of deterministic duration called *phases*. This feature allows a reconfiguration strategy to belong to the MPS[18] class for which a modeling and evaluation methodology exist [Mura 01], supported by the DEEM[19] tool [Bondavalli 04b].

Parameters

Different studies were performed on the modeled system at varying several parameters; the relevant parameters are the following:

1. Mission time t.

2. Number n of replicas in the system; this parameter impacts on the recovery strategy mainly because it determines how time sub-slots for recoveries are defined. It also put constrains on the values for f and k.

3. Probability p_I of intrusion within a replica manifesting as a permanent invalid behavior; intrusions can manifest themselves as permanent omissions with probability $1 - p_I$. Parameter p_I impacts on the recovery strategy because invalid and omission failures are treated in different ways.

18 Multiple Phased System
19 DEpendability Evaluation of Multiple–phased systems

4. Detection coverage c_M of malicious behavior of a replica. Parameter c_M impacts on the recovery strategy because only detectable faults can trigger reactive recoveries.

5. Successful attack (intrusion) rate λ^a; this parameter impacts on the recovery strategy because the detection of invalid failures triggers reactive recoveries.

6. Transient omission rate λ^o; this parameter impacts on the recovery strategy because the detection of omissive leader replicas triggers reactive recoveries.

The quantitative analysis aims to evaluate how these parameters impact on the measures of interest defined in the following section.

Measures of Interest

We are interested in measuring both the system failure probability $P_F(t)$ and the system unavailability $P_U(0, t)$ at time t; moreover, we are interested in assessing the impact of leader's omission over the above mentioned measures, given that the PRRW strategy triggers a delayed reactive recovery on the omissive leader replica only.

System failure probability

The system fails at time t if at least one of the following conditions holds:

1. the number of invalid replicas gets over f;

2. the system is unavailable for an interval of time longer then T_O.

Let $P_{FI}(t)$ be the probability of the system being failed at time t because of condition 1, given that it was correctly functioning at time $t = 0$. Let $P_{FO}(t)$ be the probability of the system being failed at time t because of condition 2, given that it was correctly functioning at time $t = 0$. System failure probability $P_F(t)$ is defined as the probability of the system being failed at time t, given that it was not failed at time $t = 0$; system failure probability is hence obtained as:

$$P_F(t) = P_{FI}(t) + P_{FO}(t). \tag{3.1}$$

System unavailability

The system is unavailable at time t if at least one of the following conditions holds:

1. the number of correct replicas is less than $f + 1$ (quorums cannot be reached);

2. there are more than $f + 1$ correct replicas, but the leader is omitting (legal messages are not forwarded).

Let $T_{\mathrm{U}}(0,t)$ be the total time the system is not failed but unavailable within $[0,\ t]$ because of one of the above conditions. Let $T_{\mathrm{A}}(0,t)$ be the total time the system is not failed within $[0,\ t]$. System unavailability $P_{\mathrm{U}}(0,t)$ is defined as the probability of the system being unavailable within $T_{\mathrm{A}}(0,t)$, given that it was correctly working at time $t=0$; system unavailability is hence obtained as:

$$P_{\mathrm{U}}(0,\ t) = \frac{T_{\mathrm{U}}(0,\ t)}{T_{\mathrm{A}}(0,\ t)}. \tag{3.2}$$

Leader contribution on system unavailability

The leader replica - beyond its role of system replica - has the special task of forwarding legal messages toward the LAN; the impact of leader's omission over the system measures of interest is hence based only on the omission about its special task. Let $T_{\mathrm{UL}}(0,t)$ be the total time the system is not failed but unavailable within $[0,t]$ because of leader's omission. The contribution of leader's omission over system unavailability, denoted with $P_{\mathrm{UL}}(0,t)$, is obtained as:

$$P_{\mathrm{UL}}(0,\ t) = \frac{T_{\mathrm{UL}}(0,\ t)}{T_{\mathrm{A}}(0,\ t)}. \tag{3.3}$$

The PRRW Model

The model presented in this section is an extension of the model proposed and evaluated in [Daidone 08]; the main differences are the following: i) this model triggers an immediate reactive recovery action as soon as the invalid behavior is detected, instead of waiting for the beginning of a recovery sub-slot;[20] ii) this model corrects a bug of the original PRRW strategy, where the detection of an omissive leader was not triggering the election of a new leader[21].

DSPN

The modeling formalism used to define the PRRW model is the DSPN[22]. DSPN models extend GSPN[23] and SRN[24], allowing for the exact modeling of events having deterministic occurrence times. Using such a formalism and associated features, the treatment of the dependencies among phases is moved from the low level of the Markov chains to the more abstract, easier to handle level of the DSPN. A DEEM model may thus include immediate transitions (represented by a thin line), transitions with exponentially distributed firing times (represented by empty rectangles), and transitions with deterministic firing times (represented by filled rectangles). Moreover arbitrary functions of the model marking may be employed to define i) firing times (rates or deterministic times) of timed transitions, ii) probabilities associated with immediate

20 This modification leads the model to better represent the reality.

21 The election of a new leader was triggered in the original PRRW by the fact that the current leader was recovering; the bug here appears when the leader is detected omissive and hence a delayed recovery is booked, but a new leader is elected only when the delayed recovery is actually started (which could happen some sub–slots after the detection).

22 Deterministic and Stochastic Petri Net

23 Generalized Stochastic Petri Nets

24 Stochastic Reward Nets

Table 3.1: Model parameters, their default values and their description.

Name	Default Value	Meaning
t	2628	Mission time (sec)
n	4	Number of replicas in the system
k	1	Maximum number of replicas recovering simultaneously
f	1	Maximum number of corrupted replicas tolerated by the system
T_{D}	146	Time duration of a recovery operation (sec)
T_{O}	60	Duration of system omission before considering the system failed (sec)
λ_i^{c}	[1.9E-7, 3.8E-7]	Crash rate of replica i (sec). Each replica has a diverse crash rate (from 1 per 60 days to 1 per 30 days)
λ_i^{o}	[1.9E-6, 3.8E-6]	Transient omission rate of replica i (sec). Each replica has a diverse rate (from 1 per 6 days to 1 per 3 days)
λ^{eo}	3.3E-2	Omission duration rate of a replica (sec). A transient omission lasts for 30 seconds (on average)
λ_i^{a}	[5.8E-5, 1.2E-5]	Successful attack (intrusion) rate of replica i (sec). Each replica has a diverse rate (from 5 per day to 1 per day)
p_{I}	0.5	Probability of intrusion within a replica manifesting as a permanent invalid behavior (if $p_{\mathrm{I}} = 0$ all intrusions manifest as permanent omissions)
c_{M}	0.7	Probability of detecting malicious behavior of a replica (if $c_{\mathrm{M}} = 1$ all detectable events are actually detected)

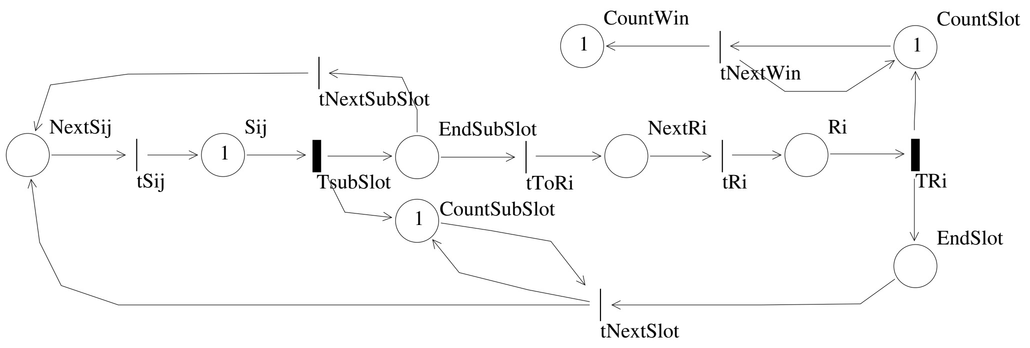


Figure 3.6: The phase net of the PRRW model; the phase net models the scheduling of the recovery sub–slots.

transitions, iii) enabling conditions (named guards) of the transitions, iv) arc multiplicities, and v) rewards.

The model is split into two logically distinct sub–nets: the Phase Net (PhN) representing the schedule of the various phases, each one of deterministic duration, and the System Net (SN) representing the behavior of the system. Each net is made dependent on the others by marking–dependent predicates that modify transition rates, enabling conditions, reward rates etc.

Reward measures are defined as Boolean expressions, functions of the net marking. Both the analytic [Mura 01] and simulation solutions [Moretto 04] can be used in order to exercise the models; the measures of interest defined in our quantitative analysis were evaluated by simulation. All the parameters (and their values) cited during the description of the model are collected and described in Table 3.1.

Phase Net (PhN)

The phase net (figure 3.6) models the PRRW scheduling shown in figure 3.5. The deterministic transitions *TsubSlot* and *TRi* model the times to perform the sub–slots S_{ij} (those grouped in the A group) and R_i, respectively. Place *Sij* contains a token during the sub–slots belonging to group A (a-periodic recovery phase) and *Ri* contains a token during sub–slot R_i (periodic recovery phase). The marking of *CountSubSlot* counts the number of the current recovery subslot (S_{ij}) within the current recovery slot. The marking of *CountSlot* counts the number of the current recovery slot within the current cycle. The marking of *CountWin* counts the number of the current cycle. The immediate transition *tNextSlot* fires when a periodic recovery slot ends, resetting the marking of *CountSubSlot* to 1. The immediate transition *tNextWin* fires when a new cycle is started, resetting the marking of *CountSlot* to 1. The immediate transitions of the phase net have priority lower than the priorities of the immediate transitions of the system net.

System Net (SN)

The system net of the PRRW model is composed by $n \leqslant 6$ similar subnets (one subnet for each replica), a subnet to keep track of system failures and

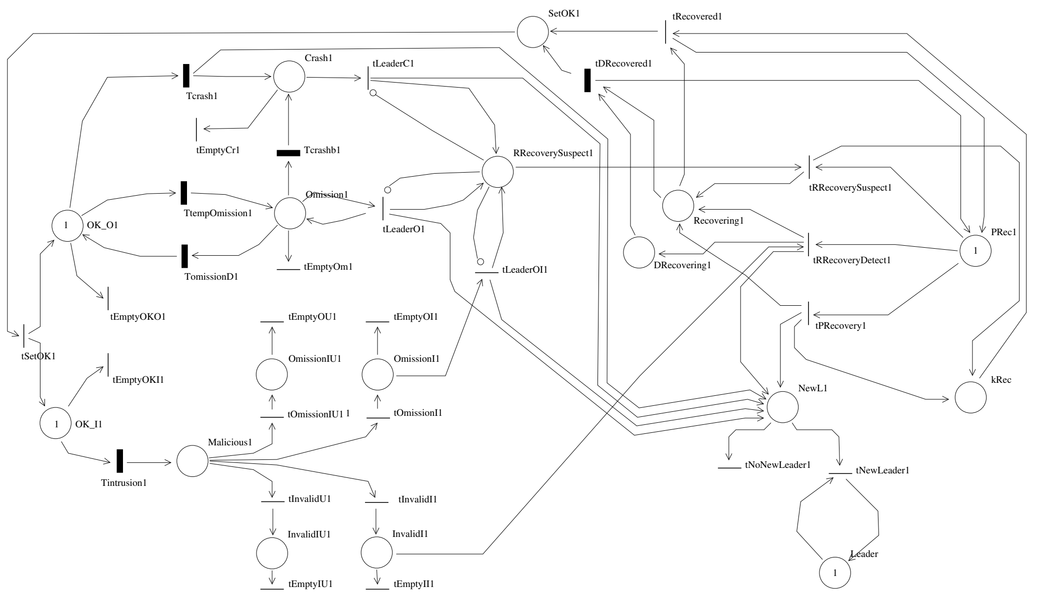


Figure 3.7: The subnet of the system net modeling failures and recoveries of replica 1 and the election of the leader (lower right part).

a subnet to initialize the model (the description of this last subnet is omitted without affecting the comprehension of the model).

Figure 3.7 shows the subnet modeling failures and recoveries of replica 1. The left–most part of the subnet models the replica failures, while the right–most part of the subnet models the replica recovery and the leader election. Places which name ends with digit "1" model replica 1, while the other places (*Leader* and *kRec*) are shared by all the sub–nets associated with the other replicas.

Replica failures

Replica failures are modeled as follows. As long as both *OK_O1* and *OK_I1* contain one token each, replica 1 is correctly working. One token in places *Crash1* or *Omission1* represents the crash of the replica or an omissive behavior as a consequence of a transient omission, respectively. The exponential transitions *Tcrash1* and *Tcrashb1* represent the time to the crash with rate λ_1^c; when the replica crashes, place *OK_I1* is emptied (the replica cannot be intruded any more). *TtempOmission1* represents the time to a transient omission exponentially distributed with rate λ_1^o. A transient omission disappears after a time modeled by the exponential transition *TomissionD1* with rate λ^{eo}.

The exponential transition *Tintrusion1* represents the time to intrusion with rate λ_1^a; the effect of the intrusion is modeled by the following immediate transitions (enabled in the same marking) and the associated places:

- *TomissionIU1* for an undetectable omission failure, with probability $(1 - c_M)(1 - p_I)$,
- *TomissionI1* for a detectable omission failure, with probability $c_M(1 - p_I)$,

- *TinvalidIU1* for an undetectable invalid failure, with probability $(1-c_\mathrm{M})p_\mathrm{I}$,
- *TinvalidI1* for a detectable invalid failure, with probability $c_\mathrm{M}p_\mathrm{I}$,

where p_I and c_M are the probability of an intrusion manifesting as a permanent invalid behavior and the detection coverage of malicious behavior, respectively.

Replica recoveries

The replica recovery is modeled as follows. Place *PRec1* contains a token as long as replica 1 is not recovering, while place *Recovering1* contains one token as long as the replica is recovering. Place *DRecovering1* contains a token during an immediate reactive recovery. Place *kRec* is used to count the number of replicas currently recovering. Place *RRecoverySuspect1* contains a token if a crash, an omission or a malicious omission occurs.

Recoveries are triggered by one of the following immediate transitions (ordered by increasing priorities): *tRRecoverySuspect1* (delayed reactive recovery triggered by suspects), *tRRecoveryDetect1* (immediate reactive recovery triggered by detections) or *tPRecovery1* (proactive recovery). The immediate transition *tRRecoverySuspect1* fires if a new a-periodic recovery sub-slot is starting (*NextSij* contains a token) and less than k replicas are recovering (*kRec* contains less than k tokens) and the replica is not going to be proactively recovered in the next periodic slot (the index of the replica is not in the interval $[(Mark(CountSlot)-1)k+1, Mark(CountSlot)k])$. The immediate transition *tRRecoveryDetect1* fires independently from the marking of the phase net. The immediate transition *tPRecovery1* fires if a periodic recovery slot is starting (*NextRi* contains a token) and less than k replicas are recovering (*kRec* contains less than k tokens) and the index of the replica is in the interval $[(Mark(CountSlot)-1)k+1, Mark(CountSlot)k]$.

When a recovery action starts, all the immediate transitions which name starts with *tEmpty* fire, emptying the following places: *OK_O1*, *OK_I1*, *Crash1*, *Omission1*, *InvalidIU1*, *InvalidI1*, *OmissionIU1*, *OmissionI1* and *OKLeadO* (*OKLeadO* will be presented hereafter). When the recovery action ends, the immediate transitions *tRecovered1* or *tDRecovered1* fire, resetting the replica subnet.

The election of the leader replica is managed as follows. The marking of place *Leader* corresponds to the index of the current leader; one token is added in place *NewL1* when one of the following events occurs: replica 1 is going to be recovered (periodic or "immediate reactive recovery), replica 1 is crashed, replica 1 is the current leader and is detected to be omissive (either benign or malicious omission). *tNewLeader1* fires if replica 1 is the current leader, triggering the mechanism of election of a new leader, otherwise *tNoNewLeader* fires. The arc from place *Leader* to place *tNewLeader1* has multiplicity equal to $Mark(Leader)$, while the arc from place *tNewLeader1* to place *Leader* has multiplicity equal to the index of the replica that will be elected as the new leader. The new leader should be the last (not crashed) replica proactively recovered, that is replica with index $j = ((n+(Mark(CountSlot)-2)k) \bmod n)+k$. If replica j is currently crashed, the next attempt is made on replica $j-1$, until a not crashed replica is found.

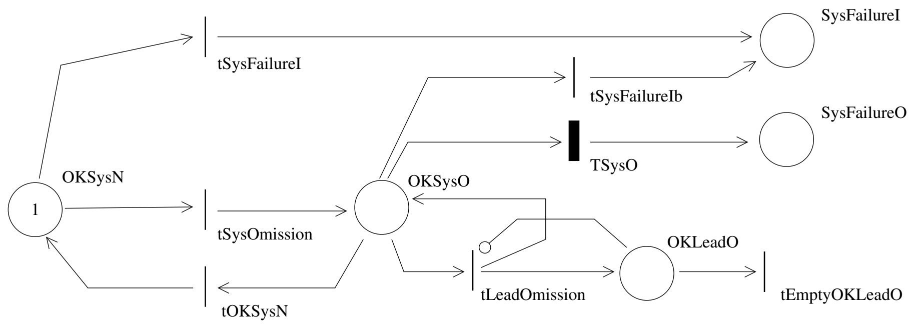


Figure 3.8: The subnet of the system net modeling the system failure.

The subnet shown in figure 3.8 models the system failure. Place *OKSysN* contains a token as long as the system is not failed and it is not omitting (there are more than f correct replicas and the leader is not crashed or omitting). Place *OKSysO* contains a token when the system is not failed but it is omitting. Place *OKLeadO* contains a token when the system is not failed, but it is omitting because the leader replica is omitting. Place *SysFailureI* contains a token when the system is failed because of invalid behavior (there are at least $f+1$ invalid replicas). Place *SysFailureO* contains a token when the system is failed because the resource unavailability lasted for an unacceptable period of time represented by the exponential transition *TSysO* with rate $1/T_{\mathrm{O}}$.

Different priorities are associated with the immediate transitions of SN, when no probabilistic choices are required. For example, all the immediate transitions of replica i have priorities lower than those of replica j, if $i < j$.

Reward Structures The evaluation of the measures of interest in DEEM involves specifying a performance (reward) variable and determining a reward structure for the performance variable, i.e. a reward structure which associates reward rates with state occupancies and reward impulses with state transitions [Sanders 91].

$P_F(t)$, $P_{FI}(t)$, $P_{FO}(t)$ The measures of interest related to system failure probability ($P_{\mathrm{F}}(t)$, $P_{\mathrm{FI}}(t)$ and $P_{\mathrm{FO}}(t)$) were evaluated in terms of three "instant of time" performance variables based on the following reward structures respectively:

```
if (Mark(OKSysO)=0 and Mark(OKSysN)=0) then (1) else (0)
if (Mark(SysFailureI)=1) then (1) else (0)
if (Mark(SysFailureO)=1) then (1) else (0)
```

$P_U(0,t)$ System unavailability $P_{\mathrm{U}}(0,t)$ was evaluated as $P_{\mathrm{U}}(0,\,t) = \frac{T_{\mathrm{U}}(0,t)}{T_{\mathrm{A}}(0,t)}$. $T_{\mathrm{U}}(0,t)$ was evaluated defining an "interval of time" performance variable which reward structure is the following:

```
if (Mark(OKSysO)=1) then (1) else (0)
```

$T_A(0,t)$ was evaluated defining an "interval of time" performance variable which reward structure is the following:

```
if (Mark(OKSysO)=1 or Mark(OKSysN)=1) then (1) else (0)
```

$P_{UL}(0,t)$

The contribution of leader's omission over system unavailability $P_{UL}(0,t)$ was evaluated as $P_{UL}(0,\, t) = \frac{T_{UL}(0,t)}{T_A(0,t)}$.
$T_{UL}(0,t)$ was evaluated defining an "interval of time" performance variable which reward structure is the following:

```
if (Mark(OKLeadO)=1) then (1) else (0)
```

Model Evaluation and System Analysis

This section presents the results of the evaluation of the measures of interest performed mainly on the PRRW model presented above and on some variants modeling the PRRW variants considered. The measures of interest were evaluated by simulation [Moretto 04] with a confidence level of 95% and a half–length confidence interval of 1%.

Model parameters

All the model parameters and the default values used for the evaluations are shown in Table 3.1 (pag. 68); the relevant parameters are described in the following:

1. Mission time t. This is the time during which the system is exercised since it starts working. t varies in [2628, 42048] sec.
2. Number n of system replicas in the system, maximum number f of corrupted replicas tolerated by the system itself and maximum number k of system replicas recovering simultaneously, with $n = 2f + 1 + k$.
3. Time duration T_D of a recovery action; the value assigned to T_D derives from experiments described in [Sousa 07].
4. Time duration T_O of system omission before considering the system failed; the value assigned to T_O derives from the temporal requirements of the secondary control in the power grid.
5. Probability p_I of intrusion within a replica manifesting as a permanent invalid behavior. p_I varies in [0, 1].

   This parameter is used to quantify how much the dependability measures vary based on the behavior of intrusions. In fact, if $p_I = 0$ then all intrusions manifest as a permanent omissive behavior; in this case, only delayed reactive recoveries (on the leader replica) can be triggered. If instead $p_I = 1$ then all intrusions manifest as a permanent invalid behavior;

in this case, intrusions on each replica can only trigger immediate reactive recoveries.

6. Detection coverage c_M of malicious behavior of a replica. c_M is the probability of detecting an intruded replica, and hence the probability of reactively recovering an intruded replica. c_M varies in [0, 1].

   This parameter is used to quantify how much the dependability measures vary based on the capability of detecting/diagnosing faults. If in fact $c_M = 0$ then no intrusions are detected; in this case, all intrusions are treated by proactive recoveries and reactive recoveries are only triggered by crash or communication omissions. If instead $c_M = 1$ then all intrusions are detected and treated by reactive recoveries.

7. Successful attack (intrusion) rate λ^a. The basic value used for the rate is $\lambda^a = 1.2E\text{-}5$; each replica has a diverse rate obtained as the basic rate multiplied by some values depending on the index replica: e.g. replica 1 has rate $\lambda^a_1 = 5\lambda^a$, replica 2 has rate $\lambda^a_2 = 4\lambda^a$, etc.

8. Transient omission rate λ^o. The basic value used for the rate is $\lambda^o = 1E\text{-}6$; each replica has a diverse rate obtained as the basic rate multiplied by some values depending on the index replica: e.g. replica 1 has rate $\lambda^o_1 = 1.93\lambda^o$, replica 2 has rate $\lambda^o_2 = 2.31\lambda^o$, etc.

The basic system configuration

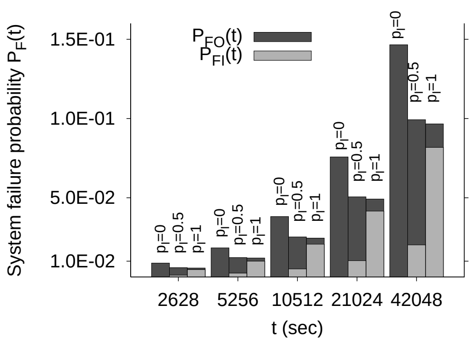
Several studies were conducted, setting the values for the parameters using this approach: a *basic system configuration* was selected and used as starting point for varying different sets of parameters for each study (e.g. a study varying only time t, another study varying only the number of replicas n). The basic system configuration follows the PRRW strategy presented in section 3.4.2, studied at time $t = 2628$, encompasses $n = 4$ replicas with probability of intrusion set to $p_I = 0.5$ and detection coverage set to $c_M = 0.7$; the fault rate are those listed in table 3.1.

Study at varying mission time t

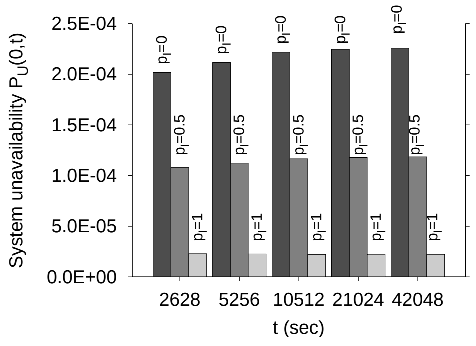
A first study was performed observing both system failure probability $P_F(t)$ and system unavailability $P_U(0,t)$ over mission time t for three different values of p_I. This study shows that $P_F(t)$ increases over time as a geometric random variable, whilst $P_U(0,t)$ seems to have an upper bound.

$P_F(t)$

Figure 3.9a shows how $P_{FI}(t)$ and $P_{FO}(t)$ change over mission time t, with $P_F(t) = P_{FI}(t) + P_{FO}(t)$. $P_F(t)$ increases exponentially over time for all the values of p_I; $P_F(t)$ behaves in fact like a geometric random variable for the following reasons: system failure probability is not null during each recovery period (cycle); the system is rejuvenated after each cycle, so we can assume that the system failure probability during the following cycle is the same as the previous one. System failure probability $P_F(t)$ cumulates hence over the recovery periods as a geometric random variable.



(a) System failure probability $P_F(t)$



(b) System unavailability $P_U(0, t)$

Figure 3.9: System failure probability $P_F(t)$ and system unavailability $P_U(0, t)$ over mission time t for different values of p_I.

The values of $P_F(t)$ are over 0.01 because of the very pessimistic values assigned to the system parameters. As p_I varies from 0 to 1, $P_F(t)$ increases of about 30% for low values of t and increases of about 17% for high values of t. For $p_I = 0$, $p_I = 0.5$ and $p_I = 1$ the value of $P_{FI}(t)$ is about 0%, 17% and 50% of the value of $P_F(t)$, respectively, independently on the values of t.

If $p_I = 0$ then $P_{FI}(t) = 0$, because there is no invalid behavior, and hence $P_F(t) = P_{FO}(t)$. As p_I varies from 0 to 1, $P_{FO}(t)$ changes from 100% of $P_F(t)$ to 17% of $P_F(t)$; the number of intrusions does not change, but the effect of intrusions changes. In fact, the value of $P_{FO}(t)$ depends on the time during which replicas are unavailable, which for $p_I = 0$ is given by the sum of the following durations:

- the time spent waiting for a delayed reactive recovery of the omissive leader;
- the time spent during the recovery on the omissive leader;
- the time spent waiting for proactive recoveries of (not leader) omissive replicas;
- the time spent for proactive recoveries (not varying for the different values of p_I).

If $p_I = 1$ then the time during which replicas are unavailable is given by the sum of the following durations:

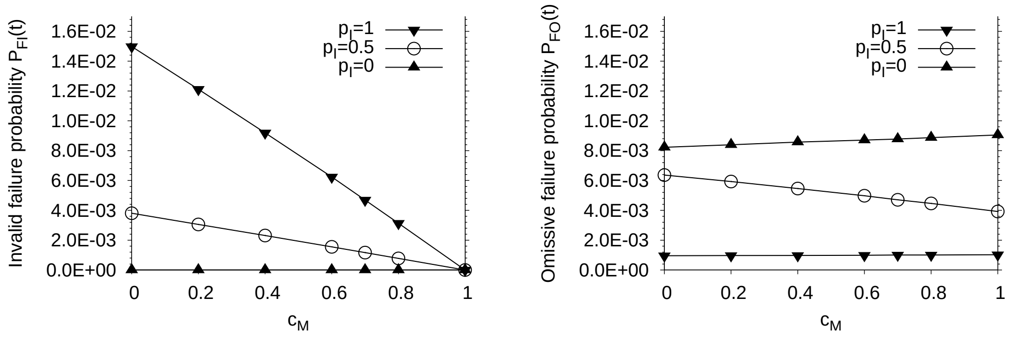
- the time spent during immediate reactive recoveries on replicas detected as intruded; the number of these recoveries is about n times the number of delayed reactive recoveries performed for $p_I = 0$;
- the time spent for proactive recoveries.

Therefore, the value of $P_{FO}(t)$ for $p_I = 1$ mainly represents the impact of recoveries (both proactive and reactive) on $P_F(t)$ (crashes and transient omissions are still present, but have lower rates than intrusions). The value of $P_{FO}(t)$ for $p_I = 1$ shows that the impact of recoveries on $P_F(t)$ is low (about 17%).

$P_U(0,t)$ Figure 3.9b shows how $P_U(0,t)$ changes over mission time t. $P_U(0,t)$ seems to have an upper bound, although $P_U(0,t)$ increases over time for all the values of p_I: for $p_I = 0.5$ and $p_I = 1$ the value of $P_U(0,t)$ is about 53% and 10% of the value of $P_U(0,t)$ for $p_I = 0$, respectively, independently on the values of t.

The trend of $P_U(0,t)$ for varying p_I is similar to the trend of $P_{FO}(t)$ shown in figure 3.9a; for $p_I = 1$ the value of $P_U(0,t)$ is mainly due to the recoveries, for $p_I = 0$ and $p_I = 0.5$ the value of $P_U(0,t)$ is negatively affected by the fact that the number of recoveries decreases but the number of omission increases.

Study at varying detection coverage Another study was devoted to evaluate both system failure probability $P_F(t)$ and system unavailability $P_U(0,t)$ at varying both the detection coverage c_M and the probability p_I of intrusions manifesting as invalid behavior. This study shows how reactive recoveries improve the measures of interest with regard to treating intrusions with proactive recoveries only.



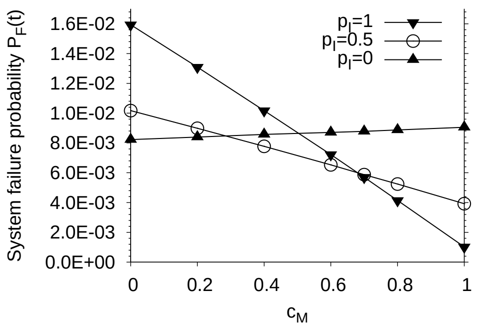
(a) Invalid failure probability $P_{FI}(t)$ (b) Omissive failure probability $P_{FO}(t)$

Figure 3.10: Impact of detection coverage c_M on both $P_{FI}(t)$ and $P_{FO}(t)$ for different values of p_I.

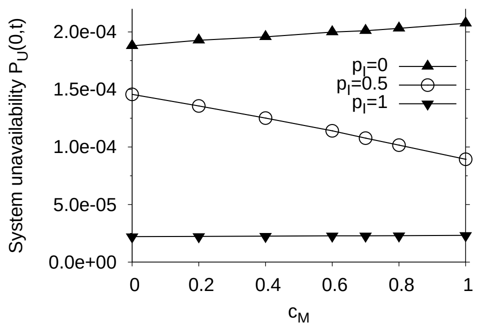
Figures 3.10a and 3.10b show how $P_{FI}(t)$ and $P_{FO}(t)$, respectively, change over detection coverage c_M for different values of p_I; in order to make easier their comparison, the same scale for the y–axis is used.

$P_{FI}(t)$ $P_{FI}(t)$ decreases as c_M increases from 0 to 1 for all the values of p_I. $P_{FI}(t)$ takes larger values for $p_I = 1$ than for $p_I = 0$. If $p_I = 0$ then the values of $P_{FI}(t)$ for different values of c_M are 0 and are not shown in figure 3.10a. $P_{FI}(t)$ takes the smallest values for $p_I = 0.2$ and is almost constant. The curve corresponding to $p_I = 1$ decreases quicker than the other curves (it decreases for about one order of magnitude) as c_M increases.

$P_{FO}(t)$ $P_{FO}(t)$ shows a different behavior with respect to $P_{FI}(t)$, given that $P_{FO}(t)$ takes larger values for lower values of p_I. $P_{FO}(t)$ is almost constant for p_I=1 (the value



(a) System failure probability $P_F(t)$



(b) System unavailability $P_U(0, t)$

Figure 3.11: Impact of detection coverage c_M on system failure probability $P_F(t)$ and system unavailability $P_U(0, t)$ for different values of p_I.

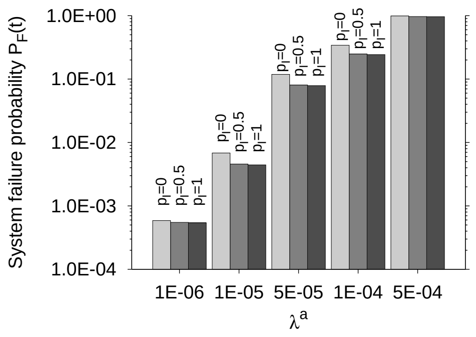
for c_M=1 is about 6% larger than the value for c_M=0); it decreases for p_I=0.5 as c_M increases from 0 to 1 (the value for c_M=1 is about 40% of the value for c_M=0); it slightly increases for p_I=0 as c_M increases from 0 to 1 (the value for c_M=1 is about 10% larger than the value for c_M=0).

The values of $P_{FI}(t)$ and $P_{FO}(t)$ for $c_M = 0$ correspond to the system configuration in which all the intrusions are treated only by proactive recoveries. The difference between the values of $P_{FI}(t)$ (and $P_{FO}(t)$) for $c_M = 0$ and $c_M = 1$ is due to the effect of treating all the intrusions by reactive recoveries: $P_{FI}(t)$ decreases, because invalid replicas reactively recovered are no longer weakening the system; $P_{FO}(t)$ is almost constant, because there are k "extra" replicas which contribute to system operation while the intruded replicas are recovering. The overall effect, shown in figure 3.11a, is that, when most of the intrusions behave as invalid ($p_I \geqslant 0.5$), system failure probability $P_F(t)$ decreases as detection coverage c_M increases. On the contrary, when most of the intrusions behave as omissions ($p_I < 0.5$), the impact of c_M on $P_F(t)$ is negligible. This stresses that, in order to reduce system failure probability (that is improve the value of $P_F(t)$), it is useful to trigger reactive recoveries and hence to set the value for c_M as higher as possible.

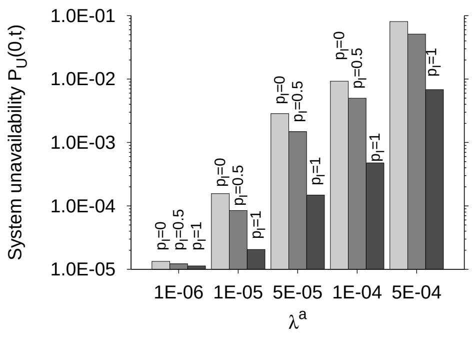
Figure 3.11b shows how system unavailability $P_U(0, t)$ changes over detection coverage c_M for different values of p_I. The trend of $P_U(0, t)$ at varying c_M is similar to the trend of $P_{FO}(t)$ shown in figure 3.10b. Again, $P_U(0, t)$ takes larger values for lower values of p_I. $P_U(0, t)$ is almost constant for p_I=1 (the value for c_M=1 is about 4% larger than the value for c_M=0); it decreases for p_I=0.5 as c_M increases from 0 to 1 (the value for c_M=1 is about 40% of the value for c_M=0); it slightly increases for p_I=0.5 as c_M increases from 0 to 1 (the value for c_M=1 is about 10% larger than the value for c_M=0). $P_U(0, t)$

The results of this study show that increasing the detection coverage of intrusions c_M has positive effects on system failure probability $P_F(t)$, particularly when the invalid behavior is dominant, and has no negative effect on system unavailability $P_U(0,t)$.

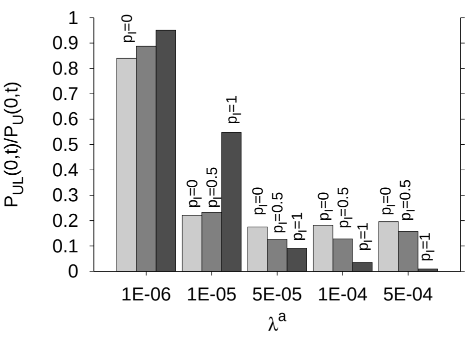
Study at varying intrusion rate A study was devoted to evaluate the impact of successful attack (intrusion) rate λ^a over system failure probability $P_F(t)$ and system unavailability $P_U(0,t)$ for different values of p_I; the percentage impact of leader omissions over system unavailability was also evaluated. The system configuration evaluated in this study is the basic system configuration where λ^a varies in {1E-6, 1E-5, 5E-5, 1E-4, 5E-4} and p_I varies in {0, 0.5, 1}.



(a) System failure probability $P_F(t)$ (y-axis is log-scale)



(b) System unavailability $P_U(0,t)$ (y-axis is log-scale)



(c) Impact of leader omissions on system unavailability $P_U(0,t)$

Figure 3.12: Impact of attack (intrusion) rate λ^a over system failure probability $P_F(t)$ and system unavailability $P_U(0,t)$ (with the impact of leader omissions) for different values of p_I.

$P_F(t)$ and $P_U(0,t)$ Figures 3.12a and 3.12b show that both system failure probability $P_F(t)$ and system unavailability $P_U(0,t)$ increase exponentially as attack rate λ^a increases (the y–axis of both figures uses a log scale); in particular the increment is about four orders of magnitudes for both the measures of interest. The behavior of the two measures of interest with respect to varying the p_I is in general the following: the value of the measure of interest decreases as p_I increases.

In particular, looking at the values of $P_F(t)$ for the smallest λ^a, figure 3.12a shows that the values for $p_I = 1$ and $p_I = 0.5$ are, respectively, 92% and 98% of the value for $p_I = 0$. The above percentages have the following trend for varying λ^a: the values of $P_F(t)$ for $p_I = 1$ are 92%, 65%, 67%, 71% and 97% of the values of $P_F(t)$ for $p_I = 0$; the values of $P_F(t)$ for $p_I = 0.5$ are about 98% of those of $P_F(t)$ for $p_I = 0$.

Looking at the values of $P_U(0, t)$ for the smallest λ^a, figure 3.12b shows that the values for $p_I = 0$ and $p_I = 0.5$ are, respectively, 85% and 93% of the value for $p_I = 1$. The above percentages have the following trend for varying λ^a: 85%, 13%, 5%, 5% and 8% if $p_I = 0$, 93%, 24%, 10%, 10% and 13% if $p_I = 0.5$.

Figure 3.12c plots the percentage impact of leader omissions $P_{UL}(0, t)$ on system unavailability $P_U(0, t)$; the impact of the leader omission decreases as successful attack (intrusion) rate λ^a increases. The shape of $P_{UL}(0, t)$ for varying p_I changes as successful attack (intrusion) rate λ^a increases: for lower values of λ^a $P_{UL}(0, t)$ has the largest value if $p_I = 0$, whilst the opposite happens for larger values of λ^a.

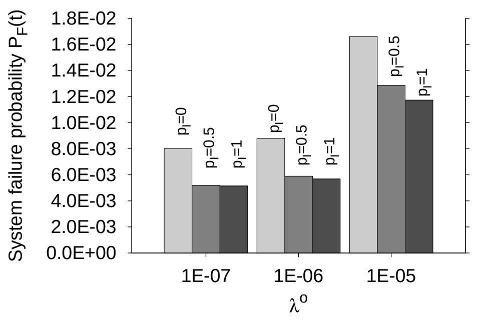
Figures 3.12a and 3.12b confirm the intuition that the attack rate deeply impacts on the system measures of interest and that the larger number of reactive recoveries triggered for increasing values of p_I positively impact on both the measures of interest. Figure 3.12c confirm that for increasing attack rate the impact of leader omission on system unavailability decreases, and hence that the main cause of system omission is the incapability of reaching quorums.

Study at varying omission rate

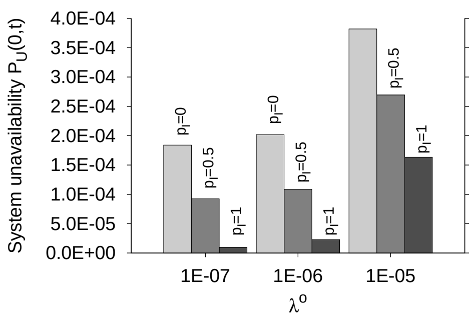
A study was devoted to evaluate the impact of omission rate λ^o over system failure probability $P_F(t)$ and system unavailability $P_U(0, t)$ for different values of p_I; the percentage impact of leader omissions over system unavailability was also evaluated. The system configuration evaluated in this study is the basic system configuration where the λ^o varies in {1E-7, 1E-6, 1E-5} and p_I varies in {0, 0.5, 1}. This study aims to better understand the impact of performing delayed reactive recoveries on the leader replica only.

Figures 3.13a and 3.13b show that in general both the measures of interest increase as the omission rate λ^o increases. The mean increment for $P_F(t)$ when the omission rate changes from λ^o = 1E-7 to λ^o = 1E-6 is about 9%, whilst the increment when the omission rate changes from λ^o = 1E-6 to λ^o = 1E-5 is about 88%. The mean increment for $P_U(0, t)$ when the omission rate changes from λ^o = 1E-7 to λ^o = 1E-6 is about 10%, whilst the increment when the omission rate changes from λ^o = 1E-6 to λ^o = 1E-5 is about 89%.

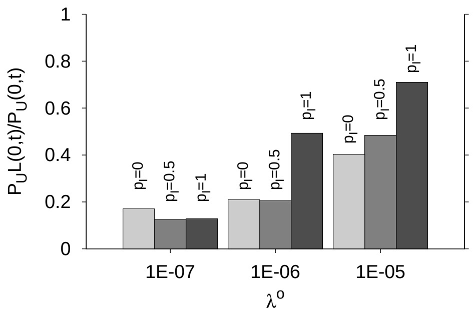
It is worth to recall that this study was performed using the default setting for the successful attack (intrusion) rate λ^a (see table 3.1), that is a value of the order of 1E-5; this means that for one of the scenarios evaluated (the right–most one in the figures) the amount of not malicious omissions is quite the same of those of malicious attacks, which can manifest as omissions based on the value of p_I. The behavior of the two measures of interest with respect to varying the



(a) System failure probability $P_F(t)$



(b) System unavailability $P_U(0,t)$



(c) Impact of leader omissions on system unavailability $P_U(0,t)$

Figure 3.13: Impact of omission rate λ^o over system failure probability $P_F(t)$ and system unavailability $P_U(0,t)$ (with the impact of leader omissions) for different values of p_I.

p_I is in general the following: the value of the measure of interest decreases as p_I increases.

Figure 3.13c shows the impact $P_{UL}(0,t)$ of leader omission over system unavailability $P_U(0,t)$; the impact of leader omission increases as λ^o increases, spanning from 17%, 21% and 40% for $p_I = 0$ to 13%, 49% and 71% for $p_I = 1$. The omissive leader is the only omissive replica to be reactively recovered, so it seems that the more omissions there are, the less is the benefit of delayed reactive recoveries on the leader. We argue this is due to the fact that a replica can be omissive due to communication errors, and there is no distinction between the detection of an omissions caused by the network and an omissions due to the replica itself: in both cases a delayed replica recovery is requested. On the contrary in such cases the replica recovery worsen the situation because it lasts on average for a longer time ($T_D = 146$ sec vs. $\lambda^{eo} = 30$ sec).

Study at varying the number of replicas

Another study was devoted to evaluate the impact of the number of replicas on both system failure probability $P_F(t)$ and system unavailability $P_U(0,t)$. The system configuration evaluated in this study is the basic system configuration where n varies in {4, 5, 6} (and hence f and k vary accordingly as described hereafter), and time is set to $t = 10512$.

When dealing with the number of replicas in the system, three parameters are relevant: n, the overall number of replicas in the system, f, the maximum number of corrupted replicas tolerated by the system and k, the maximum number of replicas simultaneously recovering without endangering the availability of the system, with $n = 2f + 1 + k$.

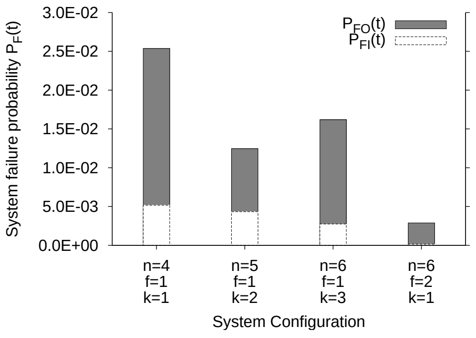
The following system configurations were evaluated:

C_1: $n = 4, f = 1, k = 1$, which is the configuration requiring the minimum number of system replicas still letting the system simultaneously recover a replica and tolerate one failed replica.

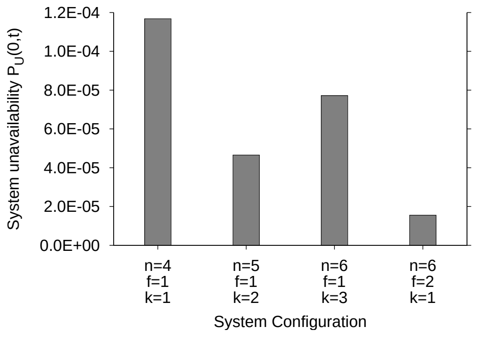
C_2: $n = 5, f = 1, k = 2$, which is the same as the previous one, but with one replica more; this configuration lets the system tolerate one faulty replica while recovering up to 2 other replicas.

C_3: $n = 6, f = 1, k = 3$, which is the same as the previous one, but with 6 replicas; this configuration lets the system tolerate one faulty replica while recovering up to 3 other replicas.

C_4: $n = 6, f = 2, k = 1$, which is the configuration requiring the minimum number of system replicas to tolerate up to two simultaneous faulty replicas and recover another replica.



(a) System failure probability $P_F(t)$



(b) System unavailability $P_U(0, t)$

Figure 3.14: System failure probability $P_F(t)$ and system unavailability $P_U(0, t)$ for different system configurations ($t = 10512$).

Figures 3.14a and 3.14b show system failure probability $P_F(t)$ (decomposed in $P_{FI}(t)$ and $P_{FO}(t)$) and system unavailability $P_U(0, t)$ for the system configurations described above.

$P_{FI}(t)$ and $P_{FO}(t)$

$P_{FI}(t)$ decreases as n (and k) increases; the trend of $P_F(t)$ is mainly due to the trend of $P_{FO}(t)$. For the same value of $n = 6$ (configuration C_3 and C_4), the higher is f and the lower is $P_F(t)$ (this behavior is shown both for $P_{FI}(t)$ and $P_{FO}(t)$); configuration C_4, although having a lower value for k, shows a lower value for $P_{FI}(t)$ because it has a more robust intrusion tolerance schema ($f = 2$); $P_{FO}(t)$ is lower because the frequency of proactive recoveries is lower ($k = 1$).

$P_U(0,t)$ The trend of $P_U(0,t)$ is quite the same of the trend of $P_{FO}(t)$ shown in figure 3.14a.

We suppose that the increment of the value of $P_{FO}(t)$ changing from configuration C_2 to C_3 is due to the combined effect of a larger number of failures (n varies from 5 to 6, but $f = 1$) and a higher frequency for proactive recoveries (k varies from 2 to 3). This supposition is supported also by comparing C_3 with C_4: the two configurations share the same number of system replicas (and hence the same number of failures), but C_4 shows a more robust intrusion tolerant schema due to a higher value of f and a lower impact on omissions due to a lower value for k. It turns out that for the setting used the lower values for $P_F(t)$ and $P_U(0,t)$ are obtained for the system configuration C_4, i.e. for higher values of f, independently of k.

Study at varying the strategy and the number of replicas The last study was performed by comparing the PRRW strategy and some other recovery strategies (which are PRRW variants) at varying the number of replicas in the system. This study aims to better evaluate the role of the different recovery actions over the measures of interest (system failure probability $P_F(t)$ and system unavailability $P_U(0,t)$).

The following recovery strategies were defined and evaluated:

$P+R_i+R_d$: this is exactly the PRRW strategy, where the following recovery actions are performed: proactive (P), reactive immediate (R_i) and reactive delayed (R_d). The recovery actions are triggered based on the rationale presented in section 3.4.2. It is worth recalling that PRRW strategy triggers delayed recovery actions on the leader replica only.

R_i+R_d: this recovery strategy triggers only immediate and delayed recovery actions (R_i and R_d respectively); proactive recovery actions are not performed. Delayed recovery actions are performed during recovery sub–slots disciplined as in PRRW. This strategy does not trigger proactive recovery actions.

$P+R_i$: this strategy triggers only proactive (P) and immediate reactive (R_i) recoveries as disciplined as in PRRW; delayed recoveries on the leader replica are not triggered.

$P+R_i^*$: this strategy is identical to the PRRW strategy except for the triggering of the recovery actions on the omissive leader: this strategy triggers an immediate reactive recovery, whilst the PRRW strategy triggers a delayed reactive recovery in this case. This strategy hence triggers proactive recoveries (P) on all replicas (based on the rationale presented in section 3.4.2) and immediate reactive recoveries both on replicas detected of being compromised, and on leader replicas suspected of being omissive (R_i^*).

The system configurations evaluated were the same as those evaluated in the study presented above, i.e. the following:

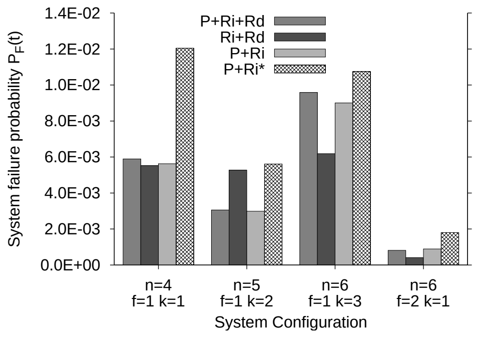
C_1: $n = 4, f = 1, k = 1$.

C_2: $n = 5, f = 1, k = 2$.

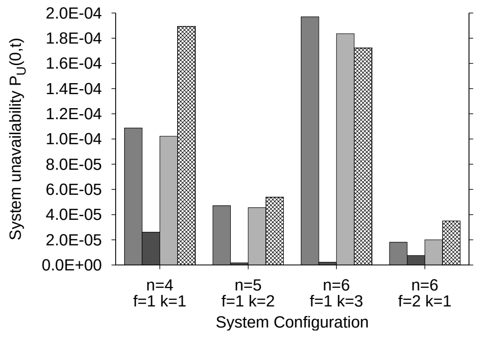
C_3: $n = 6, f = 1, k = 3$.

C_4: $n = 6, f = 2, k = 1$.

The other parameters had the same values as those assumed in the basic system configuration, in particular $c_M = 0.7$ and $p_I = 0.5$ (all the details in Table 3.1 at pag. 68).



(a) System failure probability $P_F(t)$



(b) System unavailability $P_U(0, t)$

Figure 3.15: System failure probability $P_F(t)$ and system unavailability $P_U(0, t)$ at varying the number of replicas for different recovery strategies.

Figures 3.15a and 3.15b show the values of system failure probability $P_F(t)$ and system unavailability $P_U(0, t)$ at varying the number of system replicas for all the recovery strategies evaluated.

$P_F(t)$

Figure 3.15a shows that the general trend for each reconfiguration strategy is that $P_F(t)$ decreases as n (and k) increases; the only exception to this general trend is represented by configuration C_3. Configuration C_3 shows in fact, for each reconfiguration strategy considered, values largest than those assumed for other system configurations.

The general trend when comparing the different recovery strategies for the same system configuration is the following: P+R_i+R_d and P+R_i are quite similar in all configurations, P+R_i^* has largest values than all the other strategies in all configurations, whilst R_i+R_d has the lowest values than all the other strategies but in C_2 (where it has quite the same value as P+R_i^*).

$P_U(0,t)$ Figure 3.15b shows quite the same general trends as those observed in figure 3.15a: relevant exceptions are the following: i) R_i+R_d shows very small values in all configurations; ii) the relationships among the values obtained in configuration C_3 are different, because P+R_i^* is lower than P+R_i+R_d.

The exception of configuration C_3 both in figure 3.15a and 3.15b is explained by comparing it both with C_2 and with C_4, following the same rationale discussed in the study at varying the number of replicas presented above.

Reactive delayed recoveries on the omissive leader show to be not effective on the measures of interest for the scenarios evaluated: this is demonstrated by the fact P+R_i+R_d and P+R_i shows quite the same values in all the configurations evaluated, with P+R_i showing even slightly lower values than P+R_i+R_d. When the recoveries on the omissive leader are performed immediately (as in P+R_i^*), the values for $P_F(t)$ increase in all configurations (in some cases they are even doubled). We argue that increasing the accuracy of the diagnosis of omission faults could reduce the penalty due to reactive recovery actions performed on a correct leader showing omissive behavior because of network omissions (the network omissions lasts for $\lambda^{eo} = 30$ seconds on the average, whilst the replica recovery lasts for $T_D = 146$ seconds).

Proactive recoveries show to be effective, but too many proactive recoveries show to have some negative side effects. This is evident when comparing configurations C_1, C_2 and C_3 in all the configurations which involve proactive recoveries (all but R_i+R_d), so that the best trade-off is for $k = 2$. This is evident also by observing that in all the above configurations R_i+R_d shows quite the same value for $P_F(t)$. For example, the value of system unavailability $P_U(0,t)$ for R_i+R_d is 24% of the corrsponding value for P+R_i+R_d, showing that proactive recovery actions play a relevant role in negatively impacting on system unavailability (there are many recovery actions performed on correct replicas).

Discussion about the PRRW Strategy

The CIS intrusion tolerance is currently obtained through a recovery strategy (PRRW) based on a combination of proactive and reactive recoveries. The use of both proactive and reactive recoveries shows to be effective since the two techniques possess complementary characteristics.

Proactive recoveries Proactive recoveries periodically rejuvenate all the replicas, without any need of fault detection mechanisms (also latent/hidden faults are treated). The period of the proactive recoveries defines a bounded temporal window (between two recoveries of the same replica) which represents a time limit for an attack attempt to be successful. In fact, this is the time an attacker has for conquering a majority of the replicas and thus for taking the control of the entire CIS. On the other hand, being the proactive recovery an "unconditional" recovery, the proactive recovery is applied also to correct replicas, which hence become unavailable for the time necessary to perform the recovery. Moreover, if only proactive recovery is used in a system, a replica hit by a fault will be unavailable until the end of its next proactive recovery.

Reactive recoveries

On the contrary, a reactive recovery is triggered only when a fault of a replica is detected, so its effectiveness depends both on the assumed fault model and on the coverage of the detection/diagnostic mechanism used (latent/hidden faults are not treated). Reactive recoveries of the faulty replicas contribute to decrease system failure probability, as shown in figure 3.11a; they are in fact performed as soon as possible, however within the duration of $\lceil f/k \rceil T_D$, without waiting the next periodic recovery on the same replica. In this way, the recovery and the rejuvenation of a faulty replica is anticipated with respect to its next proactive recovery, so the (faulty) replica becomes active and correct earlier.

This behavior apparently suggests that the more reactive recoveries are performed, the worse is system availability, as it appears evidently in figure 3.11b for $p_I = 1$. In this case, all the intrusions manifest as invalid behavior and all the detected intrusions trigger a reactive recovery. In reality, what happens is that the system ability to survive gets increased, whereas for low values of the coverage (thus less number of reactive recoveries) the system fails as soon as replicas get affected by faults.

Treatment of omissions vs. invalids

The PRRW strategy, as our analysis reveals, makes a significant difference in the way omission and invalid behaviors are treated. This is made evident by observing all the curves at varying values for p_I. Actually, invalid behaviors are detected with coverage c_M and trigger a reactive recovery, whereas omissive behaviors are essentially not detected: only the omission of the leader is detected and triggers some action, whilst the omissions of the followers are removed only with the proactive recovery. Increasing the capability to detect (and quickly react) to omissive behaviors is a way to improve the overall fault tolerance strategy.

3.4.4 *Direction for Improvements/Refinements*

This section identifies the directions for refining and improving the recovery strategy. An extended fault model is introduced and some modifications to the recovery schemes are presented.

New Extended Fault Model

The reactive recovery of the PRRW strategy is based on distinguishing and detecting a limited set of faults in replicas, amongst those possible to occur. Obviously, the remainder faults are treated, thanks to the strategy of proactive recoveries. We analyze this situation, under the light of the evaluation just performed, and enumerate a possible set of additional faults to be taken into account, in the sense of improving both system dependability and availability.

Faults detected by PRRW

In the PRRW strategy, the correct replicas detect the following faults:

LEADER BENIGN FAULT (LBF): The faulty leader omits to send a signed message to the LAN. A correct replica will suspect the leader to be "silent" after O_t consecutive leader omissions on the same signed message.

REPLICA MALICIOUS FAULT (RMF): The faulty replica (being it either the leader or a follower) sends an unsigned message to the LAN; a correct replica will immediately detect the faulty replica to be a "malicious sender".

It comes out that the PRRW schema takes into account both omissive and malicious faults in the leader replica, but only malicious faults in the follower replicas. The idea is that if a follower is going to have an omissive behavior, the problem will be eventually treated either by the proactive recovery or by the election of the replica as a leader (the replica will be extensively monitored in this case). In both cases, the negative effects of the faults will be eventually eliminated.

Additional faults to be detected

An additional set of faults might be considered by the current reactive recovery mechanisms, since detecting such faults and treating them using reactive recoveries would improve both dependability and availability of the system. These faults are listed below:

MALICIOUS APPROVAL (MA): a faulty replica approves an illegal message; the faulty replica is intruded, because all correct replicas verify the same security policy.

OMITTED APPROVAL (OA): a faulty replica omits to approve a legal message; the omission could be caused by communication problems (the replica never received the legal message), but it could also be the effect of an intrusion.

MALICIOUS SUSPECT (MS): a faulty replica signals the wormhole an accusation about a correct replica; the faulty replica is intruded, because a correct replica does not show any incorrect behavior.

OMITTED SUSPECT (OS): a faulty replica does not signal the wormhole any accusation about a faulty replica; the omission could be caused by communication problems (the replica never received the legal message), but it could also be the effect of an intrusion.

In the MA and MS cases, the faulty replica is intruded, so it needs to be recovered as soon as possible; if the faulty replica is not detected as such, it is still considered correct. In the OA and OS cases, faults could be caused either by communication omissions (no recovery is useful to solve the problem) or as an effect of intrusions manifesting as omissive behavior (a recovery could solve the problem). Devising the adequate mechanisms for faithful detection is a subject of further study, but we underline possible avenues in the next section.

Architectural Modifications for the Detection of the Extended set of Faults

This section describes the architecture modifications necessary to detect the faults described in section 3.4.4 and trigger the reactive recoveries. In order to perform the detection of the above faults it is necessary to allow each payload replica to be informed about all the approval results and manifested suspects taken by all the other payload replicas.

Use a shared virtual memory

A SVM[25] mechanism [Nitzberg 91, Morin 97] can be implemented as a reliable repository where each replica posts all its approval results and suspects; a majority of correct replicas is thus able to identify which replicas took the wrong approval decisions (if any) or manifested the wrong suspect (if any).

Approval results are stored for each incoming message in a data structure containing i) an identification for the incoming message m, ii) the approval decisions collected from all the replicas about m, iii) the final vote given by the wormhole about m. Suspects are stored in a data structure containing the suspecter(s), the suspected and the kind of suspect. This information is stored in the shared virtual memory, using it as a circular buffer in order to make room for newer information; therefore the SVM is used as a queue of dimension q. If the information to be broadcasted should be too heavy to be managed through the wormhole, some form of "compression" can be found.

How to use the SVM

Each message is identified using its IPsec/AH[26] MAC[27]. Each approval decision is stored in an array of n elements, where the i-th element represents approval result of replica i about message m:

`ACCEPT`: replica i approves m;

`REJECT`: replica i does not approve m;

`null`: no approval information still received from replica i about m;

`recovering`: replica i is currently recovering.

The final vote can be one of the following: `LEGAL`, `ILLEGAL` and `VOTING`.

How to perform fault detection

The follower payload behavior is monitored as follows. When message m comes from the WAN, each replica decides whether approving it or not, posting the final decision in the SVM. Not all the replicas will receive m in the same instant, and each replica will need some time in order to take the approval decision and post it in the repository, but a certain number of approval results about m will be available in the SVM at worst within T_{vote} time after the first post. Replicas that did not take any approval result till that moment and that were not recovering (those corresponding to the `null` array elements) will be suspected of omission (they could not have received m because of communication faults or they could have omitted maliciously). Given the final vote about

25 Shared Virtual Memory
26 Authentication Header
27 Message Authentication Code

m, all the correct replicas (i.e. all the replicas which approval result is in agreement with the final vote) will be able to identify all the faulty ones (i.e. all the replicas which approval result is in disagreement with the final vote) and suspect them as malicious faulty replicas.

3.5 THE FOREVER SERVICE

This section presents the FOREVER service ([Sousa 08, Bessani 08a]), which is a service that can be used in systems replicated with diversity and rejuvenated by periodic replica recoveries in order to sustain a fundamental assumption: not–increasing probability of intrusion over time. The FOREVER service has been recently defined in the scope of the FOREVER[28] project.

The main goal of the FOREVER service is to enhance the resilience of fault/intrusion–tolerant replicated systems (e.g. the CIS described in section 3.2) by allowing these systems to tolerate an arbitrary number of replica failures without increasing the total number of replicas. Such an ambitious goal is achieved through the combination of two important and complementary mechanisms: recovery and evolution.

Recovery and evolution

FOREVER allows an intrusion–tolerant system to recover from past malicious actions/faults, by cleaning the effects of such actions through periodic and on–demand recoveries that neutralize the effects of both undetected and detected faults and intrusions. Moreover, when FOREVER triggers a recovery of a certain replica, it not only cleans the effects of previous malicious actions/faults, but also evolves the replica, modifying the vulnerabilities that may be exploited by a malicious adversary, by applying a set of configuration diversity rules (e.g. changes O.S. access passwords, randomizes open ports, switches between different authentication methods). These rules ([Bessani 09]) are explained in section 3.5.1.

Hybrid model and architecture

In order to avoid the possibility that FOREVER itself becomes a victim of malicious attacks, a fault/intrusion–tolerant system enhanced with FOREVER should be built under a hybrid system model and architecture [Veríssimo 06a] in which the system is composed of two parts, with distinct properties and assumptions; these two parts are typically called *payload* and *wormhole* (see figure 3.16). The fault/intrusion–tolerant application (and replication library) runs in the payload part, exposed to arbitrary faults and asynchrony. The FOREVER service runs in the wormhole part, that is guaranteed to be secure and timely by construction. A more detailed description of the FOREVER service is available in [Sousa 08].

28 Fault/intrusiOn REmoVal through Evolution & Recovery

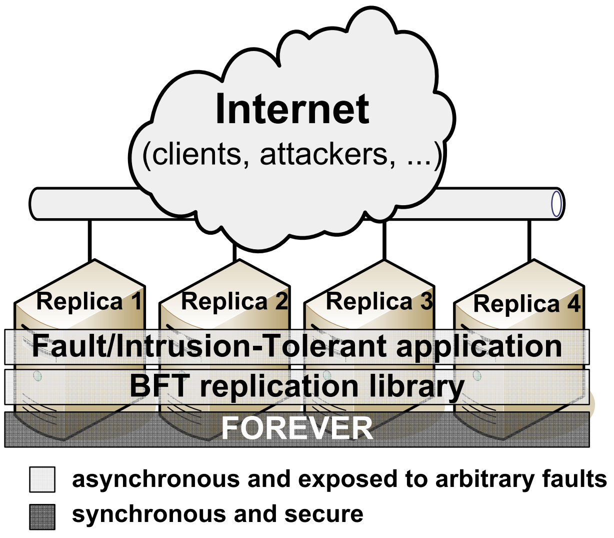

Internet
(clients, attackers, ...)
Replica 1
Replica 2
Replica 3
Replica 4
Fault/Intrusion-Tolerant application
BFT replication library
FOREVER
asynchronous and exposed to arbitrary faults
synchronous and secure

Figure 3.16: The hybrid and replicated FOREVER architecture: the fault/intrusion–tolerant application and replication library run in the payload, the FOREVER service runs in the wormhole.

3.5.1 *Introducing Diversity*

The main resilience goal of using any redundancy in an architecture design is to minimize the probability that a failure/intrusion of one of the components will lead to a system failure/intrusion.

The designer of a fault–tolerant system can use various forms of diversity in order to obtain failure/intrusion diversity between components:

SIMPLE SEPARATION OF REDUNDANT EXECUTIONS: this is the weakest form, but it may yet tolerate some faults/intrusions. In the database research community it is well known that many bugs in complex, mature software products are "Heisenbugs" [Gray 86], i.e. they cause apparently non-deterministic failures. When a database fails, its identical copy may not fail, even with the same sequence of inputs. The reported phenomena of Heisenbugs that we are aware of concern non–malicious activity, but may also be applicable for malicious behavior (e.g., some sort of brute force attack against a system which only leads to a successful penetration under certain non-deterministic combinations of behaviors in the running system).

DESIGN DIVERSITY: the typical form of parallel redundancy for fault tolerance against design faults (either accidental or intentional); the multiple replicas of the state of a system are handled by diverse software components.

DATA DIVERSITY: for some systems there may be a natural redundancy in the input language which allows the demands to the system to be expressed in syntactically different but logically equivalent forms [Ammann 88]. A practical example is the SQL language for databases where a sequence of one or more SQL statements can be "rephrased" into a different but logically equivalent sequence to produce redundant executions (see [Gashi 06] for a recent study with SQL database servers).

CONFIGURATION DIVERSITY: this form of diversity can be seen as a special form of data diversity. Software products often come with many configuration parameters affecting for example the amount of system resources they can use (amount of RAM, CPU time), port number used for communication, authentication method etc. Given the same software product, varying these parameters between two installations can produce different implementations of the data and the operation sequences on them, and thus decrease the risk of the same bug/vulnerability being triggered in two installations of the same software.

These precautions can in principle be combined. For instance, data diversity can be used with diverse software products; diverse software products can be deployed with configuration diversity of the different software. The choice will depend on the cost (e.g. purchasing the software), maintenance costs from increased complexity, time to deployment etc.

Configuration diversity rules

In what follows we summarize rules of configuration diversity which aim to enhance the resilience of software in between recoveries (see Table 3.2). Full details can be found in [Gashi 08], where the types of attacks the rule would help in alleviating are listed.

The structure of Table 3.2 is as follows:

ID: the rule identifier (the same as in [Gashi 08] for easier traceability).

RULE NAME: a concise description of the rule.

DESIGN IMPLICATIONS: implications on the architectural design of the operating system (O.S.) or application to which the rule is applied:

IMPLEMENTATION INTRUSIVENESS: is access to the internal implementation of the O.S. or applications required to apply the rule (*white–box*) or can the rule be applied simply through the utilization of the O.S. or application's configuration parameters (*black–box*)? There might also be *gray–box* solutions for which the implementation of the rule may not need to have access to the internal implementation of the O.S. or application but it can be build on top of its API[29]. This is especially useful for proprietary operating systems and applications

29 Application Programming Interface

ID	Rule name	Design implication		Security category
		Impl. intrusiveness	Client notification required?	
1	Password change	B	Yes	C
2.1	Different authentication protocols	W, B or G	Yes	C
2.2	Different Trusted Third Parties	W, B or G	No	C and A
3	Different "factors" in n-factor authentication methods	W, B or G	Yes	C
4.1	Address Space Layout Randomization (ASLR)	W	No	I
4.1.1	Pointer obfuscation	W	No	I
4.1.2	Randomization of global variables and local variables offsets	W	No	I
4.2	Address Space Partitioning	W, B or G	No	I
4.3	Stack Frame Padding	W, B or G	No	I
4.4	Basic Block reordering	W, B or G	No	I
5.1	Instruction set randomization	W, B or G	No	I
5.2	Instruction set tagging	W, B or G	No	I
5.3	Instruction Reordering	W, B or G	No	I
6.1	Diverse Linux User IDs (UID)	W, B or G	No	I and C
7	Change IP addresses of the hosts	B	Yes	C and A
8	Changing listening port numbers	W, B or G	Yes	C and A
9	Adding or deleting non-functional code	W, B or G	No	I
10.1	Varying dynamic libraries and system calls	W	No	I
10.2	Varying unique names of system files	W	No	I
10.3	Varying magic numbers in certain files (e.g., executables)	W	No	I

Table 3.2: Configuration rules for diversifying the deployment of an O.S. or an application. Abbreviations: (W)hite–, (B)lack–, (G)ray–box; (C)onfidentiality, (I)ntegrity, (A)vailability.

for which access to the implementation is not provided. For some of the rules, depending on the application or O.S. implementation, any of these implementation types (white–, black– or gray–box) are possible; in those cases we list all (or a subset of the options) for a given rule (see [Gashi 08] for more details about the scenarios under which a given implementation solution is possible).

CLIENT NOTIFICATION REQUIRED?: should client applications of the O.S. or application to which the rule is applied be notified once the rule is applied?

SECURITY FRAMEWORK CATEGORY: under which CIA (Confidentiality, Integrity, Availability) category does the rule fall into.

How rules were generated

The configuration diversity rules were generated in the following ways:

BOTTOM–UP: exploring the implementations of operating systems (e.g. Linux and Windows) and applications (e.g. database servers as PostgreSQL, Interbase, Oracle, MS SQL), and identifying the features and interfaces that can be diversified and configured in different ways.

TOP–DOWN: exploring reported vulnerabilities (such as those reported in the U.S. National Vulnerability Database (NVD)[30]) and defining rules that can "workaround" or protect against the types of vulnerabilities and attacks listed in these sources.

LATERAL: reviewing existing literature of configuration rules for protection against malicious behavior (such as [Forrest 97]).

Which rule may be most effective at improving the security of a given system will depend on the operational and threat profile of the system. For example, attacks that exploit memory programming errors (e.g. buffer overflows) are one of today's most serious security threats; these attacks require an attacker to have an in–depth understanding of the internal details of the system being attacked, including the locations of critical data and/or code. Address obfuscation techniques such as *Address Space Layout Randomization* (listed as rule 4.1 in Table 3.2) randomize the location of program data and code each time a program is executed. It has been shown that address obfuscation can greatly reduce the probability of successful attacks [Bhatkar 05, Pucella 06]. Given that recoveries force the restart of every program in the system, they are a perfect opportunity of introducing address obfuscation both at the O.S. and application level.

30 The U.S. NVD is available at `http://nvd.nist.gov/`

3.6 QUANTITATIVE EVALUATION OF THE FOREVER SERVICE

This section presents the quantitative analysis of the FOREVER service, aiming to quantify how much this service enhances the resilience of the system in which it is implemented. The quantitative analysis evaluates the probability of system failure through variation of the following parameters:

- Time between recoveries;
- Penalty due when diversity is not applied;
- Probability of common vulnerabilities;
- Mean effectiveness of configuration diversity rules applied.

The replicated system used to assess the FOREVER service is composed by n replicas and can tolerate up to f failed replicas, with $n \geqslant 3f + 1$.

Fault assumptions

We assume a replica suffers arbitrary faults; we assume one single failure mode for a replica: *failed*. For ease of modeling, we assume that a replica, as soon as it is hit by a fault, explicitly manifests a permanent failure. Despite diversity, we assume the existence of common faults (e.g. common vulnerabilities) in pairs of replicas; in particular, we assume that, given that replica i is faulty, the same fault can affect also replica j at the same time ($\mathrm{i}, \mathrm{j} \in \{1, \ldots, n\}$). The (overall) system fails if the number of failed replicas is greater than f.

Replicas were diverse both in space (design diversity) and in time (application of diversity rules).

Diversity in the space domain

Design diversity was modeled assuming that each replica has its own failure rate λ_i^a obtained by multiplying a basic value $\lambda_a = 10^{-5}$ (about one failure per day, as in [Daidone 08]) with a replica–specific multiplier[31] obtained from the results of the NVD study reported in [Bessani 08b]. We pessimistically assumed that λ_i^a was increased by an *aging penalty* value $\delta_\lambda = 10^{-6}$ (10% of the basic failure rate) when no configuration diversity rule was applied during a recovery.

Diversity in the time domain

We modeled diversity in time domain by updating the basic replica failure rate λ_i^a after each recovery in the following way:

$$\lambda^a_{i\,(\mathrm{after})} = \lambda^a_{i\,(\mathrm{before})} + \delta_\lambda\,(1 - \delta_x)\,, \qquad (3.4)$$

where $\delta_x \in \{0, 1\}$ is an *effectiveness* parameter[32] obtained as the mean value for the effectiveness parameters of all the applied rules.

Common faults

Despite diversity, we assumed the existence of common faults (e.g. common vulnerabilities) in pairs of replicas. Common faults were modeled taking into

31 The multipliers used for this analysis were 1.0, 1.8, 1.5 and 1.9.

32 $\delta_x = 0$: the rule is not effective at all, e.g. it changes the number of a listening port that is not used; $\delta_x = 1$: the rule has the optimal effect, e.g. it changes the root password after a root password compromise.

account the conditional probability δ^{ij} that replica i is faulty, given that replica j is hit by the same fault. The basic values for δ^{ij} were set based on the results of the NVD study [Bessani 08b], assuming that a common fault between two replicas was mainly due to the exploitation of a common vulnerability between the corresponding operating systems; all details are discussed in section 3.6.

Recovery strategy The modeled recovery strategy managed proactive recoveries only (as in PBFT [Castro 02] and COCA [Zhou 02]), performed sequentially "one–at–the–time" on a round robin basis. Each recovery action lasted for $T_R = 120$ seconds[33]; a waiting time T_W took places between recoveries, so that the recovery period was $T_P = n(T_R + T_W)$ seconds. The recovery process was assumed fault–free (a replica is correct after its recovery).

Measure of interest and relevant parameters

We evaluated the (overall) system failure probability $P_F(t)$, that is the probability of having more than f failed replicas, varying the following parameters:

1. Mission time t. We are interested in investigating how system failure probability changes over time.

2. Recovery period T_P, acting on the waiting time T_W between the recovery of replica i and the recovery of replica $i+1$.

3. Basic value δ_λ for the penalty on replica failure rate after a recovery action if diversity is not applied.

4. Probability δ_{ij} of common faults among different replicas; values assigned to δ_{ij} have a direct impact on system failure probability, given that using $n = 4$ replicas (so being able to tolerate at most $f = 1$ failure during a recovery slot) the overall system fails as soon as one common fault, affecting a pair of replicas, occurs.

5. Mean value δ_x for the effectiveness parameters of the configuration diversity rules.

The FOREVER Model

The quantitative evaluation of the FOREVER service was performed using a modeling methodology based on the following argument: recovery actions determine a change in the overall system configuration, therefore it is possible to represent the entire operational life split into different periods of deterministic duration called *phases*. This feature allows a reconfiguration strategy to belong

33 As in the analysis described in section 3.4, where a prototype of a system replicated with diversity, possible target for the FOREVER service, is presented.

to the MPS[34] class for which a modeling and evaluation methodology exists [Mura 01], supported by the DEEM tool [Bondavalli 04b].

Using DEEM, the model is split into two logically distinct sub–nets: the Phase Net (PhN) representing the schedule of the various phases, each one of deterministic duration, and the System Net (SN) representing the behavior of the system. Each net is made dependent on the other by marking-dependent predicates that modify transition rates, enabling conditions, reward rates etc. Reward measures are defined as Boolean expressions, functions of the net marking. Both the analytic [Mura 01] and simulation solutions [Moretto 04] can be used in order to exercise the models; the measures of interest defined for this analysis were evaluated using the analytic solver.

Phase Net

The phase net (figure 3.17) models the scheduling of the recovery actions described earlier in section 3.6. The deterministic transition *Waiting* models the time T_W between two consecutive recovery actions; the deterministic transition *Recovering* models the duration T_R of a recovery action, so place *StartRecovery* contains a token during a recovery action. The marking of place *CountLap* counts the number of recoveries performed since the start, and it is used to compute the index of the replica currently under recovery.

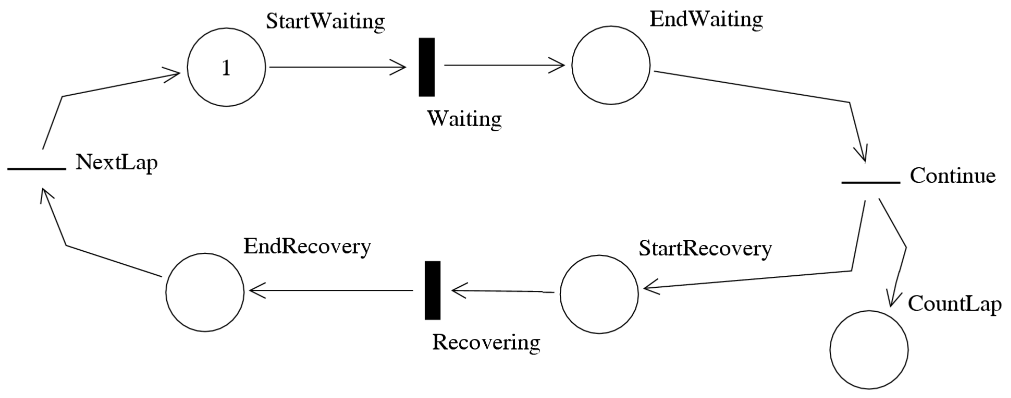


Figure 3.17: The phase net of the FOREVER model.

System Net

The system net of the FOREVER model is composed by $n = 4$ similar subnets modeling failures and recoveries of one replica each, one subnet to keep track of system failures and one subnet to model the initialization of the overall system net. The rest of the section presents the subnet modeling failures and recoveries of replica 1 (the other subnets are similar) and the subnet modeling system failure; the description of the initialization subnet can be omitted without affecting the comprehension of the model.

Figure 3.18 shows the subnet modeling replica 1; the left part of the subnet models the failure of the replica, while the right part models the replica recovery.

34 Multiple Phased System

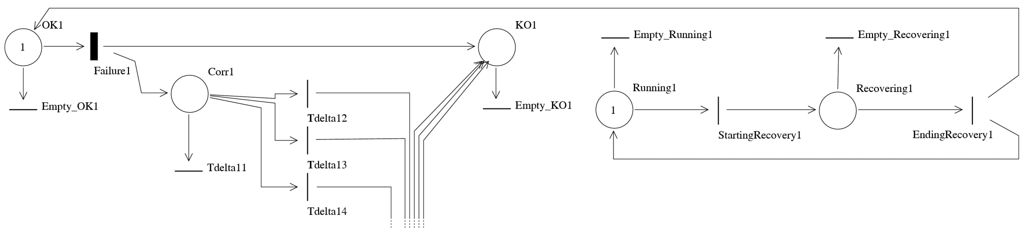


Figure 3.18: The subnet of the system net modeling failures and recoveries of replica 1.

Replica failures The failure of replica 1 is modeled as follows. As long as place *OK1* contains one token, replica 1 is correctly working; one token in place *KO1* represents the failure of the replica. The exponential transition *Failure1* represents the time to failure with rate λ_A^1 (the value of λ_A^1 changes accordingly to formula 3.4). The immediate transitions *Tdelta12*, *Tdelta13* and *Tdelta14* and *Tdelta11* fire in a concurrent way with probabilities δ_{12}, δ_{13}, δ_{14} and $(1 - \delta_{12} - \delta_{13} - \delta_{14})$ respectively. Transitions *Tdelta12*, *Tdelta13* and *Tdelta14* are connected with places *KO2*, *KO3* and *KO4* respectively (those places are not represented in figure 3.18 for space reasons) and are used to model the common faults between replica 1 and the other replicas. The rationale behind is the following. As soon as replica 1 fails, one token is put in place *Corr1*; all the four *Tdelta** transitions are now activated, but one and only one of them fires, emptying place *Corr1*. If *Tdelta11* fires then nothing happens, modeling the case in which the fault causing the failure of replica 1 is not a fault in common with another replica. If instead *Tdelta12* fires, then one token is put in place *KO2*, emulating this way a common fault between replica 1 and 2 (place *KO1* contains one token also, because of the firing of *Failure1*). Similar behavior is captured with the firing of *Tdelta13* and *Tdelta14*.

Replica recoveries The recovery of replica 1 is modeled as follows. Place *Running1* contains a token as long as replica 1 is not recovering, whilst place *Recovering1* contains a token as long as replica 1 is recovering. Recoveries are triggered based on the marking of the phase net: *StartingRecovery1* fires based on the marking of both *StartRecovery* and *CountSlot* (to discriminate whether the current round is the round in which replica 1 has to be recovered); *EndingRecovery1* fires when place *EndRecovery* contains one token (the firing priority of *EndingRecovery1* is greater then the firing priority of *NextLap*). As long as replica 1 is recovering, the immediate transitions *Empty_OK1* and *Empty_KO1* can fire, emptying the places which they are connected to.

System failure The subnet shown in figure 3.19 models the system failure. Place *SysNotFailed* contains a token as long as the system is not failed, whilst place *SysFailed* contains one token when the system fails. The immediate transition *SysFailure* fires based on the marking of the places *KO1*, *KO2*, *KO3* and *KO4* (more than $f = 1$ of these places contain one token); the occurrence of a system failure makes all the immediate transitions *Empty_** fire, stopping all the activity in the system net.

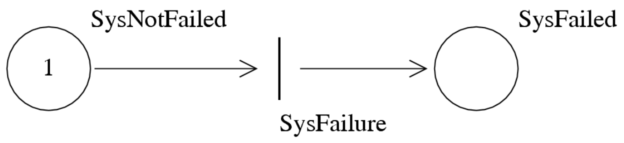


Figure 3.19: The subnet of the system net modeling the system failure.

Reward structures

The evaluation of the measure of interest $P_F(t)$ involves specifying a performance (reward) variable and determining a reward structure for the performance variable, i.e. a reward structure which associates reward rates with state occupancies and reward impulses with state transitions.

System failure probability $P_F(t)$ was evaluated in terms of an "instant of time" performance variable which is based on the following reward structure:

```
IF (MARK(SysFailed)=1) THEN (1) ELSE (0)
```

Evaluation Parameters

This section presents the definition of the evaluation parameters and the values assigned to those parameters.

We assumed to use $n = 4$ replicas, which is the minimum number of replicas in order to tolerate $f = 1$ faulty replicas. Replicas use different operating systems: FreeBSD, Solaris, Linux and Windows 2000 (shortened to "Win2K"). In order to set the values of replica failure rates and probabilities of common faults, the results of the NVD study [Bessani 08b] were used[35]. The information in [Bessani 08b] relevant for the scope of this analysis is reported in Table 3.3: the number of vulnerabilities of the above mentioned operating systems (2nd column) and the number of common vulnerabilities between all the couples of operating systems (from 3rd to 5th columns).

Replica failure rate

The values for the replica failure rates λ_A^i were set as follows. The values shown in Table 3.3 about reported vulnerabilities between 1999 and 2007 document that 1424 vulnerabilities were reported in that period, that is an average of 178 vulnerabilities per year. Given that this was the trend before 2007, we can assume the same trend after 2007, and hence consider that there are 178 new vulnerabilities every year. Given that patches are eventually defined to correct the reported vulnerabilities, and that those patches are eventually installed, we assume that the mean period during the year in which the reported vulnerabilities are *uncovered* (they exist in the system, but are not patched) is 10% of the overall year. We assume also that an attacker has the 10% of the overall knowledge about the uncovered vulnerabilities.

35 The available data encompasses vulnerabilities reported between 1999 and 2007.

O.S.	vulns	common vulns		
		Win2K	Linux	Solaris
FreeBSD	229	3	11	18
Solaris	411	3	5	
Linux	437	3		
Win2K	347			

Table 3.3: Operating Systems' reported vulnerabilities (from NVD [Bessani 08b]) between 1999 and 2007.

All the above considerations lead our replicated system to suffers about 2 successful attacks per year to the brand new vulnerabilities, which corresponds to a (basic) successful attack rate $\lambda_A = 2E\text{-}4$ (expressed wrt hours). In order to differ the failure rate of each replica, the basic failure rate was multiplied by the following values: 1.0 for FreeBSD, 1.8 for Solaris, 1.9 for Win2K and 1.5 for Linux. These multipliers were obtained based on the values in Table 3.3, weighting the number of reported vulnerabilities of a specific operating system using the total number of vulnerabilities.

Penalty for missing diversity The basic value for the penalty δ_λ on replica failure rate λ_A when no diversity is applied during recovery actions was set pessimistically to $\delta_\lambda = 2E\text{-}5$, that is 10% of the basic failure rate.

Common faults Common faults were modeled taking into account the conditional probability δ^{ij} that replica *i* is faulty, given that replica j is hit by the same fault. The basic values for δ^{ij} were set–up according to the information in Table 3.3, assuming that a common fault between two replicas is mainly due to the exploitation of a common vulnerability between the corresponding operating systems. Table 3.4 shows the values used for δ^{ij}.

δ_{ij}		j			
		FreeBSD	Solaris	Linux	Win2K
i	FreeBSD	-	0.029	0.017	0.005
	Solaris	0.029	-	0.006	0.004
	Linux	0.017	0.006	-	0.004
	Win2K	0.005	0.004	0.004	-

Table 3.4: Probability of common faults among all the couples of O.S. considered.

Effectiveness of diversity rules Twenty diversity rules R_k were considered (see Table 3.2 for a complete list of them) and applied during each recovery action. The effectiveness δ_k of a given rule depends on several factors, so it is very difficult to define its value precisely; this is the reason why we considered their mean value δ_x at their place and evaluate it for a set of values.

The recovery duration T_R was set[36] to $T_R = 120$ seconds; the duration of the waiting time between two consecutive recovery actions was set to $T_W = 0$.

Recovery duration

Study	T_W (sec)	δ_λ(hours)	δ_x	δ_{ij}
1	{0, 360, 480, 840}	0	0	δ_{ij}
2	0	{0, 2E-6, 2E-5, 1E-5}	0.8	δ_{ij}
3	0	0	0	$\{0, 1, 5, 10\} \times \delta_{ij}$
4	0	2E-5	{0, 0.2, 0.4, 0.6, 0.8, 1}	δ_{ij}

Table 3.5: Values assigned to the evaluation parameters for each study evaluated; the values for δ_{ij} are those listed in Table 3.4.

3.6.1 *Evaluation results*

We evaluated the (overall) system failure probability $P_F(t)$ (i.e. the probability of having more than f failed replicas) over mission time t, varying the following parameters: i) recovery period T_P (acting on the waiting time T_W), ii) *aging penalty* δ_λ, iii) probability δ_{ij} of common faults and iv) mean value δ_x for the effectiveness parameters of the configuration diversity rules.

The measure of interest was evaluated using the analytic solver for all the studies performed (with $\epsilon = 10^{-10}$, $Maxiter = 10^4$)[37]. Table 3.5 summarizes the values assigned to the model parameters for each study.

Study at varying the waiting time T_W

The first study was performed to find the optimal configuration for the recovery strategy described above; this means having played with the waiting time T_W between two consecutive recovery actions. System failure probability $P_F(t)$ was evaluated over time for four different recovery strategy configurations, corresponding to the following values of T_W: {0, 360, 480, 840} (sec). This study was evaluated in the simplistic case where no penalty is applied on replica failure rate ($\delta_\lambda = 0$) and no configuration diversity rules are applied during recoveries ($\delta_x = 0$) in order to isolate the impact of T_W on the measure of interest.

Figure 3.20 shows that system failure probability $P_F(t)$ increases over time for all the configurations evaluated, with a bias proportional to the value of T_W;

36 This value comes from [Daidone 08], where a prototype of a system replicated with diversity, possible target for the FOREVER service, was studied

37 ϵ represents the error tolerance, $Maxiter$ the maximum number of iterations that has to be considered by the transient solution method

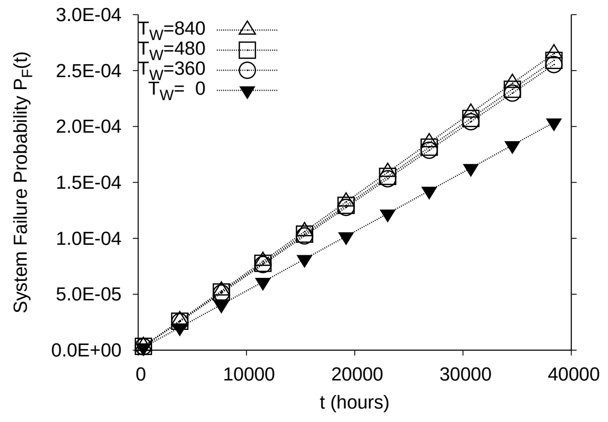


Figure 3.20: System failure probability $P_F(t)$ over mission time t at varying the waiting time T_W between two consecutive recovery actions; no penalty applied ($\delta_\lambda = 0$), no diversity applied ($\delta_x = 0$).

the best result for $P_F(t)$ is obtained for $T_W = 0$ (evidenced in figure with a black filled marker). This is in line with the qualitative perception that the more recovery actions are performed in a certain time window, the less the probability of system failure is. Based on this result, all the rest of the evaluations (described in the rest of this section) were performed by setting $T_W = 0$.

Study at varying the penalty δ_λ

The second study was performed at varying the basic penalty δ_λ on the replica failure rate, in order to quantify how much it affects system failure probability. System failure probability $P_F(t)$ was evaluated over time for three values of δ_λ {2E-6, 2E-5, 1E-5} and for the (ideal) case in which $\delta_\lambda = 0$. This study was evaluated in the case in which the mean effectiveness of the configuration diversity rules applied during replica recoveries is $\delta_x = 0.8$.

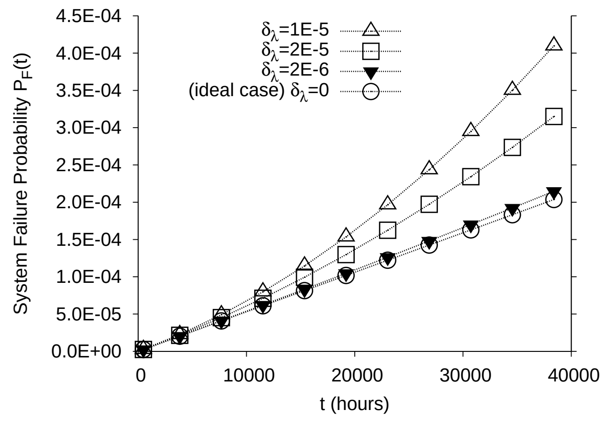


Figure 3.21: System failure probability $P_F(t)$ over mission time t at varying the "aging penalty" δ_λ (with $\delta_x = 0.8$).

Figure 3.21 shows that the impact of δ_λ on system failure probability $P_F(t)$ increases as the penalty increases, so that after 40000 hours of service, one order of magnitude on δ_λ corresponds to a doubled value for $P_F(t)$. It is worth to mention that δ_λ is not a parameter under the control of the system designer, so here we are making a pessimistic hypothesis.

Study at varying the probability δ_{ij} of common faults

The third study was performed at varying the probabilities δ_{ij} of common faults between couples of replicas. System failure probability $P_F(t)$ was evaluated over time for four set of values for δ_{ij}. The first set considered contains null values (no correlations at all), the second set is the basic one (see Table 3.4), the third set is obtained from the basic one by multiplying each value by 5, the fourth set by multiplying by 10. This study was evaluated in the simplistic case where no penalty is applied on replica failure rate ($\delta_\lambda = 0$) and no configuration diversity rules are applied during recoveries ($\delta_x = 0$) in order to isolate the impact of δ_{ij} on the measure of interest.

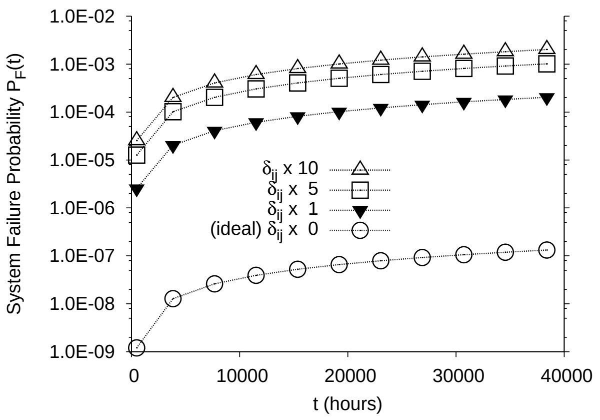


Figure 3.22: System failure probability $P_F(t)$ over time at varying the probability of common fault δ_{ij} (with $\delta_\lambda = 0$ and $\delta_x = 0$); y is log–scale.

Figure 3.22 shows how system failure probability $P_F(t)$ varies over time for each set of values for δ_{ij}; please note that y–axis is log–scale. The curve with the circle marker correspond to the (ideal) scenario in which there are no common faults between replicas (all $\delta_{ij} = 0$).

The shape of the curves does not change at varying the multiplier used, but the corresponding values change: between the ideal case (x0) and the basic δ_{ij} (x1) there are three orders of magnitude; four orders of magnitude between ideal case and worst case considered (x10).

Study at varying the mean effectiveness δ_x of diversity rules

The forth study was performed at varying the mean effectiveness δ_x of the configuration diversity rules applied during replica recoveries. System failure probability $P_F(t)$ was evaluated over time for $\delta_x \in \{0, 0.2, 0.4, 0.6, 0.8, 1\}$. The two extreme values have the following interpretation: if $\delta_x = 0$ then the effect is the same as not applying rules at all; if $\delta_x = 1$ then the effect is the same as not having the penalty δ_λ at all (ideal case). This study was evaluated in the pessimistic case in which the penalty is set to $\delta_\lambda = 2\text{E-}5$.

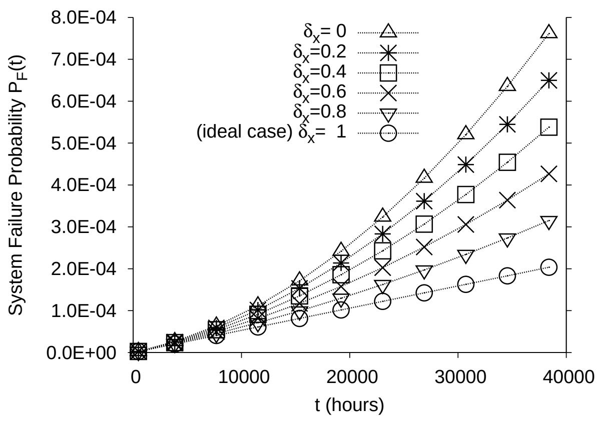


Figure 3.23: System failure probability $P_F(t)$ over mission time t at varying the mean effectiveness δ_x of the configuration diversity rules ($\delta_\lambda = 2\text{E-}5$).

Figure 3.23 shows that the impact of δ_x on system failure probability $P_F(t)$ is such that after 40000 hours of service the difference among the two extreme values corresponding to $\delta_x = 0$ and $\delta_x = 1$ is less then one order of magnitude (it is about 6E-4).

Analysis outcomes

There is always an increasing trend of the probability of system failure $P_F(t)$ for every line in every graph presented in this section, even if replicas are recovered and even if diversity rules are applied. This depends both on the system intrinsic characteristics and on the assumptions made: replicas are inevitably subjected to fail, so, even if they are recovered, they still do fail; the introduction of diversity in the time domain was assumed to, at best, maintain constant the failure rate of a replica, so the failure rate cannot decrease.

The results presented in this section lead to the following considerations.

Proactive recoveries help in keeping down system failure probability (figure 3.20); furthermore, it is better to recover as fast as possible: of course, this is true unless availability or performability measures of interest are considered (e.g. there could be a trade–off between system failure probability and system availability).

Design diversity highly improves the dependability of the system: figure 3.22 shows also the difference between cases of no failure correlation and the failure dependencies between the operating systems which were obtained from the

NVD study. The lower is the common failure rate between the replicas deployed in the system, the lower is the overall system failure rate[38]

Another result is that the higher is the effectiveness of the configuration diversity rules, the lower is the system failure probability (figure 3.23), substantiating the claim that it is better to apply some rule (even a modestly effective one) than applying no rule at all. Of course, this is only true under the assumption that rules cannot worsen security.

3.7 CONCLUDING REMARKS

The quantitative analysis described in this chapter has shown that enhancing the resilience of a redundant system is a matter of trade–offs when both system failure probability and system availability have to be taken into account.

All starts from the consideration that system failure probability inevitably increases over–time and cumulates over the recovery periods as a geometric random variable (see section 3.4.3), so the question is how to keep system failure probability down as much as possible within each cycle. This chapter has shown that periodically recovering replicas and introducing diversity both in the time and space domains are effective and complementary strategies. Some considerations have to be made.

Proactive recoveries are effective in lowering down system failure probability, and performing a periodic recovery on the same replica as soon as possible improves this benefit (as shown in 3.20), but performing too many proactive recoveries in a given time window shows to have some negative side effects on system availability (e.g. see the comparison among configurations C_1, C_2 and C_3).

Reactive recoveries help in keeping system failure probability down (see figures 3.11a, 3.12a or 3.12b), but their effectiveness strictly depends on the capability of correctly detecting and diagnosing faults: see the role of c_M and the effect of performing delayed reactive recoveries on leaders diagnosed to be omissive (see the results of the study comparing the strategies and the number of replicas).

Another important parameter to be set up is n, the number of replicas in the system. Increasing n not necessarily helps in keeping down the measures of interest, being an additional replica not only an help for contributing to the service provided by the system, but also a source of faults (see the study at varying n).

The introduction of diversity in space and time domains helps in lowering down the impact of common faults on the measures of interest: see for example

38 This is an obvious observation, but the analysis performed has given a measures of how much worse the overall system failure becomes when the common failure rates of the replicas increases.

figure 3.22 or figure 3.23 (regardless of the effectiveness of the configuration diversity rules, it is better to apply some rule than applying no rule at all).

4 SOLUTIONS FOR ADAPTIVE OPERATION

This chapter describes the application of the principles presented in chapter 2 within the scope of the HIDENETS[1] project. HIDENETS addressed the provision of available and resilient distributed applications and mobile services with critical requirements on highly dynamic and possibly unreliable open communication infrastructures.

The HIDENETS objectives and requirements are presented, focusing in particular on a practical application taking advantage of the HIDENETS results: the platooning application. The platooning application is a safety critical application requiring timeliness and security. A specific test bed (the platooning test bed) was defined within the project in order to demonstrate a prototypal implementation of the platooning application [Marques 09].

The architecture of HIDENETS [Casimiro 07] is presented, focusing in particular on the services devoted to diagnosis and reconfiguration: *diagnostic manager* and *reconfiguration manager*. Both the diagnostic manager and the reconfiguration manager are defined following the basics of the diagnosis framework proposed in section 2.2. Then the implementation of a simple service (named *DM+RecM*) performing diagnosis and reconfiguration is presented in the scope of the platooning test bed.

The last part of the chapter presents a quantitative analysis of some fault diagnosis and reconfiguration strategies for the management of a replicated server pool (part of this work is in [Lollini 08]). The focus of the quantitative evaluation is understanding the impact of the parameters of the reconfiguration strategies and obtaining the optimal parameter configuration for a selected set of fault scenarios. The implementation of such a redundancy management in HIDENETS is enclosed in the middleware service named *replication manager*.

4.1 HIDENETS OBJECTIVES AND REQUIREMENTS

The overall objective of HIDENETS was to develop the required innovative technology to enable the design and validation of applications and services in mobile scenarios that have to satisfy stringent dependability and resilience requirements. Challenges, threats, and resilience requirements were identified by thinking mainly to car-to-car scenarios with additional infrastructure support.

1 HIDENETS is a recent STREP (Specific Targeted Research Project) funded by the IST programme of the European Commission (Contract IST-2004-26979). `http://www.hidenets.aau.dk/`

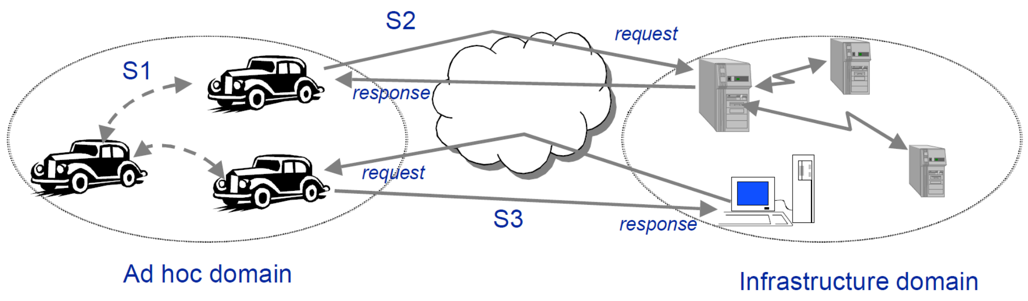


Figure 4.1: Application domains addressed in HIDENETS: the *ad–hoc domain* (e.g. a communinicating vehicular network) and the *infrastructure domain* (e.g. a back-bone IP network).

HIDENETS distinguished two fundamentally different domains (depicted in figure 4.1):

AD–HOC DOMAIN: in this domain service access and service deployment are performed in a (multi-hop) wireless setting; this implies that the properties of communication links are subjected to large variations, and that network topologies -and hence reachability relationships- change dynamically. A specific instance of this domain is the car–to–car use case scenario, where the ad–hoc domain consists of communicating vehicular networks.

INFRASTRUCTURE DOMAIN: this domain consists of a back–bone IP network connecting both service providers as well as service clients. Parts of the ad–hoc domain may be connected to the infrastructure domain via cellular access (GPRS/UMTS) or via WLAN hot-spots.

The challenges faced up by HIDENETS beyond the dynamicity and mobility of the scenarios considered were the openness of the system, the use of COTS components, the heterogeinity of the system nodes (different capabilities and available resources) and the large number of nodes involved.

Fault categories considered encompassed design–time and run–time faults, timing (omission, crash) and value faults, transient and persistent faults, with accidental and malicious causes. Detailed fault models and failure modes depend on application type and technical realization.

Several application were envisioned which could illustrate the potential benefits of the HIDENETS services from the end–to–end perspective; the focus here is on the *platooning application*, which will be described in details in section 4.3.

4.2 HIDENETS ARCHITECTURE

An Hybrid Architecture

The HIDENETS node was designed using the hybrid architecture proposed in [Veríssimo 06a] and depicted in section 2.2; two separated parts can be hence identified in the node architecture, one corresponding to the wormhole, the other one to the payload. The architectural block which realized the architectural hybridization concept in HIDENETS is the *resilience kernel* [Casimiro 07]. The resilience kernel is represented in figure 4.2 by the white box comprising both the *simple trusted services* and the *resilience hardware* boxes.

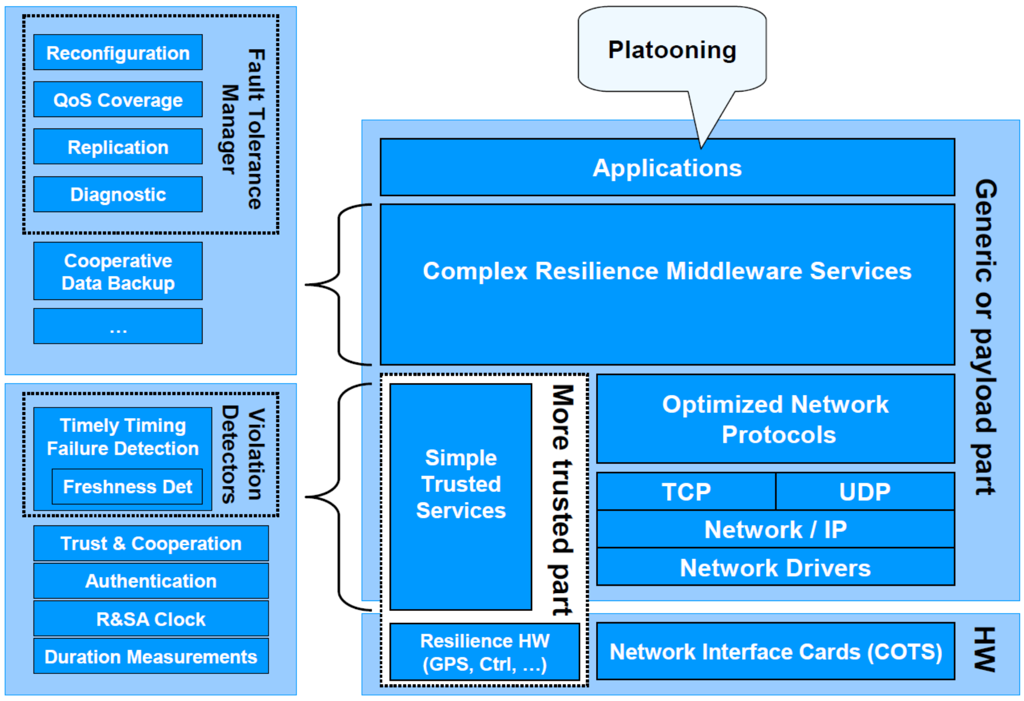


Figure 4.2: The hybrid architecture of a HIDENETS node.

The following layers (listed bottom–up) compose the architecture of a generic HIDENETS node:

THE HARDWARE LAYER: it comprises all the hardware supporting the node, part of which being COTS;

THE COMMUNICATION/NETWORKING SUPPORT: this block runs at operating system level and includes general communication functions related to OSI layers 2, 3 and 4 (e.g. TCP, IP, Network drivers) on top of which optimized network protocol are defined.

THE MIDDLEWARE LAYER: this layer provides functions supporting resilience. The software included within the middleware layer is thought of as trustworthy and is allowed to use operating system functions, read variables and even interact with low level hardware.

THE APPLICATIONS LAYER: regular applications may be installed and run by users in this layer; even if some resilience functions are built into the applications themselves, this layer is thought of as potentially untrusted.

The resilience services defined within the HIDENETS node were classified into two categories: simple and trusted vs. complex resilience services.

Simple and trusted resilience services

The simple and trusted resilience services are referred to as *timeliness and trustworthiness oracles*, where the term "oracle" was used just to underline the idea of a service that provides trusted service. Those services are enclosed in the resilience kernel and were designed as simple/small as possible, since simplicity is fundamental to ensure increased predictability and trust (according to the design philosophy of wormhole–based systems).

Complex resilience services

The complex resilience services are enclosed in the middleware, so they are running on top of a potentially asynchronous model; these services use and rely on the trusted part if and when required, but essentially for the execution of critical steps during their operation.

The focus here is on diagnostic manager and reconfiguration manager services, both enclosed in the so–called *fault tolerance manager* block (see the top–left part of figure 4.2), which is the subset of complex resilience services directly related to the management of the fault–tolerant activities.

4.2.1 *The Diagnostic Manager service*

This section summarizes the design principles of the diagnostic manager (DM) service; the full details can be found in [Casimiro 07].

The DM is the middleware service in charge of managing all the activities necessary to judge if the HIDENETS system (or parts of it) is (are) working properly or not. The DM judgments are sent to the fault removal mechanisms, which deals with how to prevent the fault to be activated again (reconfigurations in the HIDENETS system are managed by the reconfiguration manager described in section 4.2.2).

The DM, being part of the Fault Tolerance Manager block (see figure 4.2), is useful for all applications and use cases where resilience requirements are relevant. For example, considering the Platooning use case, a failed car must not flood the platoon with inconsistent information, otherwise the platoon is forced to stop.

Design organizations

The HIDENETS environment is made up by two different categories of nodes:

- Fixed nodes, which belong to the server infrastructure: they are wired interconnected and can have redundant resources;
- Mobile nodes, which belong to ad–hoc domain: they are COTS devices provided with wireless capabilities in order to be able to connect to fixed nodes or to other mobile nodes.

Nodes inside the same category can be considered quite similar, but there are significant differences when comparing a fixed node with a mobile one:

- High vs. low amount of system resources: fixed nodes can do more and faster than mobile ones (e.g. they can benefit from redundant/replicated resources, they do not have power constraints).
- Wired vs. wireless connections: different speed and stability capabilities.
- Fixed nodes are always in the system, mobile ones alternate periods during which they can connect to the HIDENETS infrastructure with periods where they can't do that, e.g. they are in an ad–hoc island (they can more easily have connection problems, low battery, etc. or they simply can be intentionally switched–off).
- Mobile devices should be reasonably cheap, to promote their diffusion.

On the basis of the specific node functionalities and capabilities, different fault classes can be identified, so it is necessary to develop specific diagnostic solutions able to assess the internal status of such different nodes/entities.

In addition, the specific characteristics of the HIDENETS nodes impose restrictions on the realization of the diagnosis service with different trade–off between accuracy and promptness: fixed nodes may afford diagnosis solutions taking longer time to gather information and emitting a judgment, while mobile nodes call for quicker, although possibly less accurate, solutions (due to both lower resources and shorter time an entity may be traceable).

Moreover, HIDENETS system is highly dynamic and distributed, so each node interacts with a dynamic group of "reachable" nodes (mobile and fixed ones) along time. It is important for each node to be able to assess the status of nodes inside its group, because problems inside a node could negatively propagate into other nodes of the group. Hence, the three different design organizations proposed in section 2.2 were used in HIDENETS to design the diagnosis activity:

1. Local Diagnosis: diagnosis performed by a node inside the same node on local resources/services (node auto–control).
2. Private Diagnosis: diagnosis performed by a node on a remote node based on the private perception of the remote node.
3. Distributed Diagnosis: diagnosis performed in a distributed way among a set of collaborating nodes when an agreement about the healthy status of each node (Byzantine resilient) is necessary.

Input from other building blocks

To accomplish its task with reference to a monitored system component, the DM mainly needs to acquire error/deviation information from error/deviation detectors related to the monitored component. In principle, any local error/deviation detection mechanism, from packet–level CRC to application–level exception handlers, is an eligible feeder of DM. Which are the relevant sources of error/deviation detection information strictly depends on which is the specific entity to be diagnosed, which are the faults it may be affected by and which are the consequences of such assumed faults. In order to make such detection information available to DM, proper interfaces toward DM have to be set up; alternatively, such signals can be conveyed and stored in a repository, accessed by DM when necessary.

The complete list of services/oracles of the HIDENETS middleware which are sources of error/deviation detection information for DM can be found in [Casimiro 07].

Output to other building blocks

Diagnosis judgments made by the DM component are used by the following HIDENETS entities:

- *Reconfiguration Manager*, which will be detailed in section 4.2.2.
- *QoS Coverage Manager*, which performs some probabilistic analysis based on diagnostic data and performance historical data.
- The maintenance center, to take the appropriate actions to physically isolate and repair the component diagnosed as faulty (when necessary).

Concerning the relationship between the DM and the applications running in the HIDENETS node, the DM was designed so that it is transparent to the applications. This means that DM does not offer functionalities directly exploitable by applications, but instead it contributes to offer to an application a more reliable HIDENETS platform to run on.

4.2.2 *The Reconfiguration Manager service*

This section summarizes the design principles of the reconfiguration manager (RecM) service; the full details can be found in [Casimiro 07].

The RecM is the middleware service managing system reconfigurations (selecting both the time of reconfiguration and the proper reconfiguration policy to be applied) and system preventive maintenance on the basis of information coming from the system (e.g. from diagnostic manager or from QoS coverage manager) and/or from application needs (e.g. applications organized in several different operational modes). The RecM aims to:

- Bring back the system to provide correct (although possibly degraded) services after the occurrence of some malfunctions.
- Properly manage system resources in order to provide the required QoS levels (within unavoidable limits) after the occurrence of some deviations from expected QoS.
- Decide the application of predefined preventive maintenance policies, possibly alerting the operator service in order to physically repair/replace

modules which fail to satisfy the maintenance tests and, in such a case, implementing the appropriate reconfiguration of the involved subsystem.

- Change system reconfiguration on the basis of operational mode required by the running application.

Reconfiguration in complex distributed systems is typically approached following a hierarchical approach [Porcarelli 04]; this is appropriate also in the context of HIDENETS, where node reconfiguration is organized at two levels:

- Reconfiguration local to a node, in order to either resist to local diagnosed faults or to better exploit local available resources.
- Reconfiguration at multi–node level, to better manage faults and resources at system level. Global reconfiguration, possibly performed inside a group of both fixed and mobile nodes, promotes efficient reorganization of system resources, taking into account higher level information on the whole involved group of components, thus overcoming the restricted vision at the single node.

The reconfiguration activity involves also the underlying communication levels, thus the RecM interacts also with the *communication adaptation manager*[2].

RecM works on–line, basically performing the following activities:

- Gathering information about the status of the entities monitored by the Diagnostic Manager.
- Gathering information about the evaluated QoS levels from the QoS Coverage Manager.
- Selecting the proper reconfiguration of components/system (if any) based on the gathered information; the choice of the reconfiguration action is guided by the expected benefit of applying it, obtained through a quantitative evaluation support. (e.g. with respect to the MTTF[3] measure, or to the time necessary to perform the reconfiguration, or to some more complex performability measures)
- Selecting the current operational mode (if any is defined) and the corresponding configuration.
- Triggering the proper actuators to put in place the selected reconfiguration.

2 The communication adaptation manager service is the HIDENETS middleware service responsible for performing cross-layer optimization.

3 Mean Time To Failure

- Triggering efficient preventive maintenance operations on the basis of an evaluation support to compare several possible alternatives (as suggested in [Porcarelli 01, Porcarelli 04]).

Static vs. Dynamic Approach

Reconfiguration is performed according to different approaches; the extremes are described in the following, but intermediate solutions are possible as well:

STATIC APPROACH: the set of envisioned reconfiguration strategies is defined at system design time; each envisioned strategy is statically associated with specific patterns of faults/deviations of a number of system components. This association is performed through a look–up table, which is accessed on–line to retrieve the appropriate reconfiguration strategy.

DYNAMIC APPROACH: many strategies are applicable for the same diagnosed scenario; the choice of the reconfiguration to be applied is performed on line through a proper evaluation support, which is fed with the specific system and environment conditions at the time the reconfiguration action is triggered by the DM subsystem. This approach is of course more accurate than the previous one, but more costly in terms of time and resources needed to perform the on–line choice.

Active vs. Passive behavior

The RecM component can be defined as a passive or an active component:

- The passive RecM is activated on demand by the system entities requiring a reconfiguration (e.g. Diagnostic Manager when specific faults are diagnosed in specific components, or QoS Coverage Manager when certain application QoS requirements are no longer satisfied).

- The active RecM is always active; it is in charge of managing the gathering of the necessary information, possibly activating/reactivating specific diagnostic components. While the active RecM collects information about diagnosed faults, it estimates whether a reconfiguration strategy has to be applied and when it has to be applied (if any). After having triggered a specific reconfiguration, the active RecM could trigger some post-reconfiguration checks in order to verify whether the desired effects were obtained (possibly collecting feedback that could be used to guide reconfiguration selections in the future).

4.3 PLATOONING TEST BED

The objective of the platooning test bed within HIDENETS was to validate a set of mechanisms that allow to detect and react to violations of timeliness requirements, and to malicious intrusions. Those mechanisms were designed to assure safety in the presence of timing uncertainty[4], to be able to dependably adapt to changing environment timings and to handle certain malicious intrusions.

Platooning Application

The application chosen within HIDENETS as the more representative for the platooning test bed was the platooning application [Egel 08]. The objective of the platooning application is to let a group of cars (the so called *platoon*) to drive on a highway as a platoon; the application has to provide both positional and velocity control of platooners[5] in order to operate safely as a platoon (that is, avoiding collisions). In order to implement the platooning application, platooners mainly need to communicate with each other via an ad–hoc network, receive GPS[6] coordinates from satellites, and use proximity sensors to identify obstacles. Figure 4.3 gives an overview of the platooning scenario.

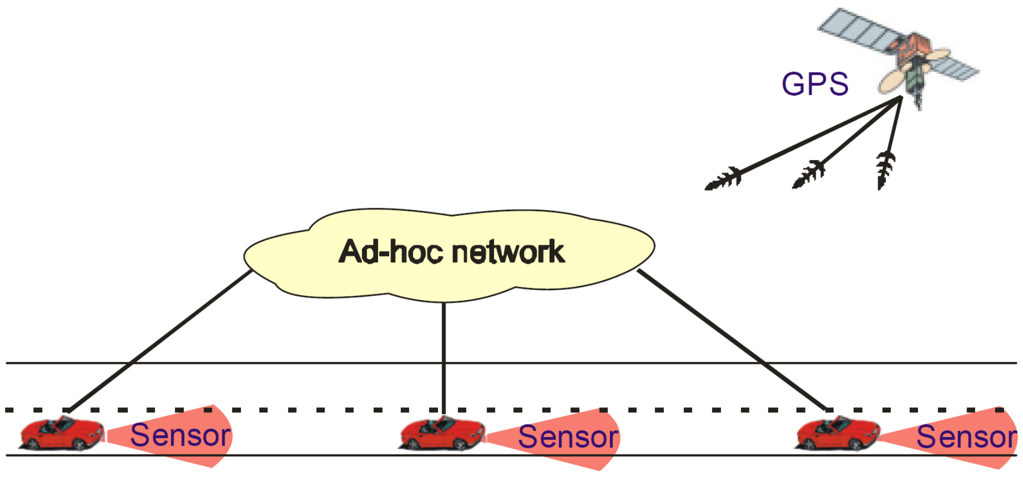


Figure 4.3: An overview of a platooning scenario, where some vehicles drive as a platoon with the help of a GPS system (for position and clock) and some proximity sensors (for detecting physical obstacles).

The platooning application has several requirements, from timeliness to security, which make it a (safety–)critical application. The main challenges posed by the platooning application are the following:

- Timely communication: if messages containing the position of a platoon member arrive too late to another platoon member (e.g. just the following one), this may result in too late reactions to a manoeuvre in the platoon, leading to safety problems (e.g. car collisions).
- Availability of reliable broadcast or multicast service: it is essential that all messages sent out by each vehicle in the platoon can reach all addressed

4 The main causes of timing errors in wireless multi–hop communication networks are communication delay and jitter.
5 Platooners are the vehicles member of a platoon.
6 Global Positioning System

platoon members; this relates to throughput performance as well as to transmission errors.

- Optimized re–distribution of available radio resources: what happens when two large platoons meet in the same geographical area and hence need to use the same radio resources? In this case, the amount of radio resources required by the two platoons increases and needs to be guaranteed for each platoon.
- Trustworthy communication: any information exchanged inside the platoon needs to be trustworthy (false messages may negatively affect safety).
- Dependable adaptation: the platooning application requires a reasonable potential for adaptation, by adjusting the behavior of the vehicles in order to cope with specific situations, including possible communication failures or simple degradation of the communication quality.

The HIDENETS architecture was provided with safety–critical services which can support the implementation of the platooning application [Marques 09].

HIDENETS implementation of the platooning scenario

Each platooner is an HIDENETS mobile node, and hence it is structured according to the HIDENETS reference architecture presented in section 4.2; figure 4.4 shows the architecture of a HIDENETS node as it was customized for the platooning test bed.

The platooning test bed assumes that vehicles move in a straight line and that there are no other obstacles except other vehicles moving ahead or behind. Moreover, vehicles are moving all in the same and common direction.

Each platooner has a number of components: i) a driver interface encompassing the accelerator, the brake and the platooning activation switch; ii) a general purpose computing device which hosts the payload and consequently the platooning decision making algorithm[7]; iii) a local wormhole, which includes a simple safety distance control sub-system that is always active whenever the car is not in platooning mode, and interfaces to several sensors and actuators (among them, GPS receivers). The payload communicates with other nodes via ad–hoc networks (based on the IEEE 802.11 standard); the payload communicates with the local wormhole through an internal car connection. The payload also hosts the middleware resilience services.

Given that both the vehicle and the environment in which the vehicle drives were emulated in HIDENETS[8], the interfaces between the wormhole and the sensors and actuators of the vehicle were emulated. Figure 4.5 shows the interface through which the vehicle can be driven (the arrows for accelerating and braking, the "start button" to start the platooning application); this interface

7 The decision making algorithm collects and processes context information about platooners, and set the resulting speed control commands to the vehicle.

8 The emulator used to emulate both platooners and environment is TORCS (The Open Racing Car Simulator); `http://torcs.sourceforge.net/`

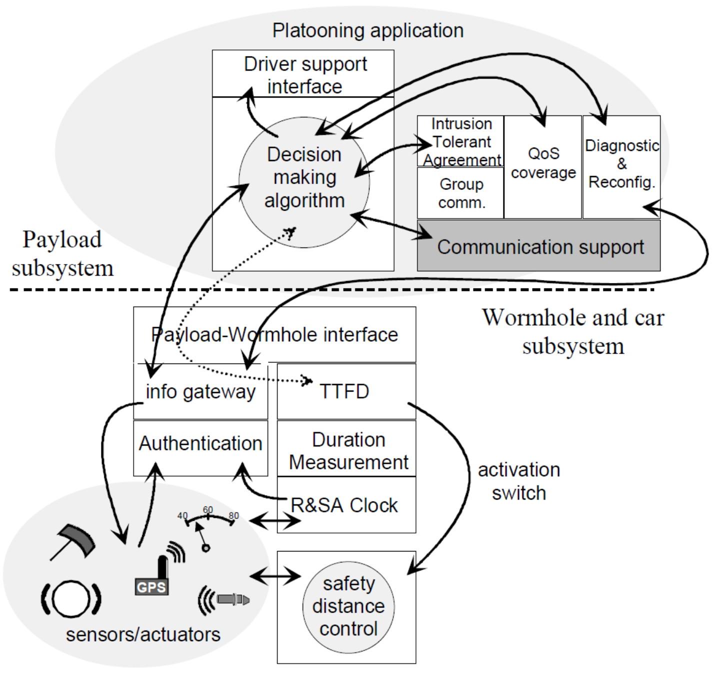


Figure 4.4: Architectural view of the HIDENETS node as customized for the platooning test bed: the wormhole, hosting the oracles, is interfaced with sensors and actuators in the vehicle; the payload hosts the platooning application and the middleware services.

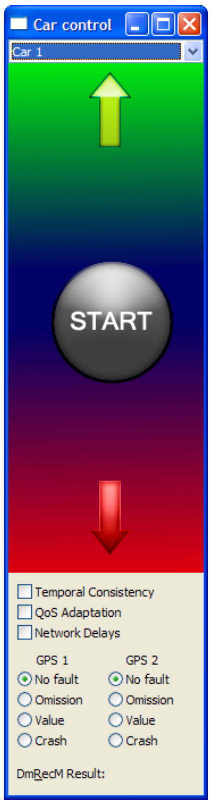
includes also some check–boxes and radio buttons used to control some environmental aspects (e.g. to inject payload timing failures, communication delays and faults in the GPS receivers). The same interface is also used to perceive the diagnosed status of the GPS receivers and the corresponding reconfiguration action in place (if any).

4.3.1 *Diagnostic and Reconfiguration Managers in Platooning test bed*

This section presents the specialization of the DM and RecM services in the ad–hoc scenario; in particular, the specialization is customized within the platooning test bed. The specialization focuses on the local diagnosis and reconfiguration aspects, where the mobile node monitors its resources and performs reconfigurations taking into account only the benefit on the node itself.

Each vehicle used in the test bed is assumed to have two GPS receivers; the DM and RecM services are used to manage the (active) redundancy of GPS receivers.

For the scope of the test bed, the DM and RecM services were enclosed together in a single service named DM+RecM. The DM+RecM service is in charge of diagnosing the GPS receivers and performing their reconfiguration;



☐ Temporal Consistency
☐ QoS Adaptation
☐ Network Delays

GPS 1	GPS 2
[✓ No fault]	[✓ No fault]
Omission	Omission
Value	Value
Crash	Crash

Figure 4.5: Interface of the emulator controlling both the driving of the emulated vehicle (top-most part with the arrows and the "start button") and some environmental aspects (bottom part including check–boxes and radio buttons).

the duplication–with–comparison technique [Siewiorek 98] was used as the underlying criteria: the DM+RecM service filters the information flows produced by the GPS receivers, dropping information generated by faulty GPS receivers and constructing a unique consolidated information flow to be made available to the decision making algorithm of the platooning application.

System description and assumptions

The vehicle running the platooning application is assumed to have two GPS receivers working in parallel, named as GPS_1 and GPS_2; the receivers can be identical or diverse. Given that both vehicles and environment were emulated in the test bed, GPS receivers were emulated also. The rationale behind the emulation of the GPS receivers follows the basics of the widespread protocols used to manage GPS receivers (in particular the NMEA0183 protocol[9]).

GPS frames Each GPS receiver is assumed to generate GPS frames containing values re-

9 The NMEA 0183 standard defines a combined electrical and data specification for communication between electronic devices (e.g. GPS receivers) and other types of instruments.

lated to the movement of the vehicle:

1. Time reference t.
2. Estimated position at time t.
3. Estimated instantaneous speed at time t.

Let $f_i(t)$ be the GPS frame generated by GPS_i at time t, and let F_1 and F_2 be the flows of GPS frames generated over time by GPS_1 and GPS_2 respectively.

Each GPS receiver is assumed to proactively generate a GPS frame every second, so each F_i is made up by subsequent frames $f_i(t)$, $f_{i+1}(t)$, etc. All the GPS receivers are directly connected to the wormhole, as shown in figure 4.4, and hence each GPS receiver proactively sends a flow of GPS frames to the wormhole.

Fault Model

The faults assumed to affect a GPS receiver are the following:

CRASH: the GPS receiver stops sending any information to the wormhole (as the GPS receiver was disconnected from the wormhole). This is a permanent fault requiring a maintenance operation.

VALUE: the GPS receiver sends wrong values to the wormhole (e.g. due to multi–path interferences caused by reflected GPS signals arriving at the GPS receiver, typically as a result of nearby structures or other reflective surfaces).

OMISSION: the GPS receiver is not able to estimate some (or all the) values which are part of a GPS frame (e.g. when the vehicle is within a tunnel). This is a transient fault; there is nothing that the DM+RecM service can do/request to treat this fault.

Failure modes

If GPS_i is not faulty, then the values contained in the GPS frames $f_i(t)$ are correct and represent a discrete time approximation of the movement of the vehicle; this means that those values satisfy some constrains related to the physical laws describing the movement of the vehicle. The constraints taken into account in the test bed are related to mean acceleration M_a, mean velocity M_v and mean displacement M_s computed for each time interval $[t, t + \delta_t]$ (with $\delta_t = 1$). These constraints will be detailed in the following text.

If GPS_i is affected by a value fault at time t, some (or all) the values contained in the GPS frame $f_i(t)$ do not satisfy the above mentioned constrains; should this happen, the $f_i(t)$ is considered a "not plausible" frame (the details about the definition of "plausible" GPS frame are given hereafter). If GPS_i is crashed at time t, no GPS frame $f_i(t)$ is sent to the wormhole. If GPS_i is omitting at time t, an empty GPS frame $f_i(t)$ is sent to the wormhole.

Diagnosis and reconfiguration

The DM+RecM service performs the following actions:

1. Diagnosis of the GPS receivers: this step is implemented by inferring the status of the GPS receivers by monitoring over time the information flows they generate.

2. Reconfiguration: this step consists in dropping information generated by faulty GPS receivers and merging the remaining GPS frames in order to produce the consolidated flow F.

Diagnosis The diagnosis of the GPS_i receiver at time t bases on the the GPS frame $f_i(t)$: if $f_i(t)$ does not exist, a crash is diagnosed. If instead $f_i(t)$ exists, an omission is diagnosed if $f_i(t)$ contains null values, whilst a value fault is diagnosed if $f_i(t)$ is not plausible.

Plausible GPS frame If the information contained within GPS frame $f_i(t)$ is assumed correct, the information contained in a subsequent GPS frame $f_i(t+1)$ is checked to be plausible based on the criteria described hereafter. Each GPS frame $f_i(t)$ contains the following values:

$$f_i(t) =< t, v(t), s(t) >$$

$v(t)$ is the velocity measured at time t, $s(t)$ is the position measured at time t. Considering two subsequent frames received from the same GPS receiver (δ_t=1), the following "intermediate" values are computed (acceleration is assumed constant):

$$\begin{aligned} \delta_v &= v(t+1) - v(t) \\ \delta_s &= s(t+1) - s(t) \\ \delta_m &= v(t)\delta_t + \tfrac{\delta_v * \delta_t}{2} \end{aligned}$$

The following checks are performed:

Mean acceleration	$m_a = \frac{\delta_v}{\delta_t} < Max_a$
Mean velocity	$m_v = \frac{\delta_s}{\delta_t} < Max_v$
Mean displacement	$\delta_s = \delta_m \pm \Delta\%$

If all the above checks are satisfied than the $f_i(t+1)$ frame is considered *plausible, not plausible* otherwise. The settings used for the Max_a, Max_v and Δ bounds are the following:

$$\begin{aligned} Max_a &= 9\tfrac{m}{s^2} \\ Max_v &= 55\tfrac{m}{s} \\ \Delta &= 1 \end{aligned}$$

Table 4.1: Reconfiguration strategy implemented by the DM+RecM service; the faulty condition encompasses crash, omission and value faults.

Reconfiguration action		Diagnosis of GPS_2 healthy	Diagnosis of GPS_2 faulty
Diagnosis of GPS_1	healthy	merge GPS_1 and GPS_2	forward GPS_1
Diagnosis of GPS_1	faulty	forward GPS_2	forward null value

The coverage of the plausibility check is not 100%.

Reconfiguration The DM+RecM performs the reconfiguration after having diagnosed the GPS receiver; the rationale behind reconfiguration is sketched in Table 4.1. If both GPS receivers are diagnosed healthy (that is, $f_1(t)$ and $f_2(t)$ are both plausible), then the output frame $f(t)$ is computed as the frame containing the following values:

$$f(t) =< t, \frac{v_1(t) + v_2(t)}{2}, \frac{s_1(t) + s_2(t)}{2} >$$

If only the GPS_i receiver is healthy at time t (either i=1 or i=2), then the output frame is $f(t)=f_i(t)$. If all GPS receivers are faulty at time t, then the output frame $f(t)$ is initialized by sing null values.

Implementation and storyboards

The DM+RecM was implemented as a C++ class and was integrated in the code used for the platooning test bed. The demonstration of the functionalities offered by the DM+RecM service was performed by defining some storyboards: faults were injected through the interface shown in figure 4.5 and the output of the DM+RecM was saved in the storyboard. Some examples of the storyboards used are shown hereafter.

Injected faults were represented using the following codes:

- `-` if no fault was injected;
- `O` if an omission was injected;
- `V` if a value fault was injected;
- `C` if a crash was injected.

The output of the DM+RecM was coded as follows:

- `OK` if the DM+RecM forwarded a proper frame;

- <!> if the DM+RecM forwarded a null value.

First storyboard The first storyboard was executed not injecting any fault, with the expectation that all the data forwarded by the DM+RecM was correct. The results reported in the storyboard shown in Table 4.2 confirmed the expectations.

Table 4.2: Storyboard 1: no faults injected.

Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
GPS_1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
GPS_2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Out	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK

Second storyboard The second storyboard planned the crash of GPS_1 at time $t = 6$, with the expectation that the DM+RecM was able to manage the redundancy of GPS receivers and forward correct data. The results reported in the storyboard shown in table 4.3 confirmed the expectations.

Table 4.3: Storyboard 2: crash of GPS_1 at time $t = 6$.

Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
GPS_1	-	-	-	-	-	C	C	C	C	C	C	C	C	C	C
GPS_2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Out	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK

Third storyboard The third storyboard planned both the crash of GPS_1 at time $t = 6$ and an omission fault in GPS_2 at times $t = 7, 8, 9$. The expectation was that the DM+RecM was able to manage the redundancy of GPS receivers and forward correct data when some data were available from the receivers. The results reported in the following storyboard (Table 4.4) confirmed the expectations.

Table 4.4: Storyboard 3: crash of GPS_1 at time $t = 6$ and omission of GPS_2 at times $t = 7, 8, 9$.

Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
GPS_1	-	-	-	-	-	C	C	C	C	C	C	C	C	C	C
GPS_2	-	-	-	-	-	-	O	O	O	-	-	-	-	-	-
Out	OK	OK	OK	OK	OK	OK	<!>	<!>	<!>	OK	OK	OK	OK	OK	OK

Fourth storyboard The fourth storyboard planned value faults in GPS_2 at times $t = 7, 8, 9, 10$. The expectation was that the DM+RecM was able to manage the redundancy of GPS receivers and forward correct data from the beginning to the end of the experiment. The results reported in the following storyboard (Table 4.5) confirmed the expectations.

Table 4.5: Storyboard 4: value faults in GPS_2 at times $t = 7, 8, 9, 10$.

Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
GPS_1	-	-	-	-	-	-	V	V	V	V	-	-	-	-	-
GPS_2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Out	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK

Fifth storyboard

The fifth storyboard planned value faults in both GPS_1 and GPS_2 starting at time $t = 4$ and lasting for the rest of the experiment. The expectation was that the DM+RecM was able to forward correct data at least until time $t = 4$, and that it was not forwarding wrong data till the end of the experiment. The results reported in the following storyboard (Table 4.6) confirmed all the expectations.

Table 4.6: Storyboard 5: value faults in both GPS_1 and GPS_2 starting at time $t = 4$.

Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
GPS_1	-	-	-	V	V	V	V	V	V	V	V	V	V	V	V
GPS_2	-	-	-	V	V	V	V	V	V	V	V	V	V	V	V
Out	OK	OK	OK	<!>	<!>	<!>	<!>	<!>	<!>	<!>	<!>	<!>	<!>	<!>	<!>

The DM+RecM demonstrated to meet all the expectations made, both in terms of promptness and accuracy. The demonstration through storyboard did not cover all the possible combinations of faults, but just some significant one.

4.4 QUANTITATIVE EVALUATION OF SERVICE ACCESS WITH REPLICATED SERVERS

One of the problems faced up in the HIDENETS project was the problem of managing a server pool in the infrastructure. In order to minimize the impact of server failures and communication failures on the clients, diagnosis and reconfiguration activities are involved: diagnosis has to detect whether a server or a communication link has failed or not (and hence diagnosis in this case is very close to failure detection), reconfiguration has to make the clients contact not failed servers of the pool.

This activity, despite involving also diagnosis and reconfiguration, was implemented in HIDENETS as part of the Replication Manager service. Replication Manager is in fact the middleware service defined to handle replication, hiding to the application both the details about the changes of a replica server and changes in the topology of the system, and trying to optimize the selection of server replicas in order to optimize the application/user experience.

This section presents a model–based quantitative evaluation of dependability and performance metrics of this client–server based service. The quantitative evaluation was applied to obtain the optimal parameter configuration for a selected set of fault scenarios; the full details of the evaluation can be found in [Lollini 08].

The service considered was implemented on top of a distributed server pool, following the major design paradigms of the RSerPool[10] architecture [Lei 08]. The scenario considered encompasses multiple clients that want to access a service provided by a set of replicated servers; the application considered as a reference was a transaction based, SIP-like[11]application [Rosenberg 02].

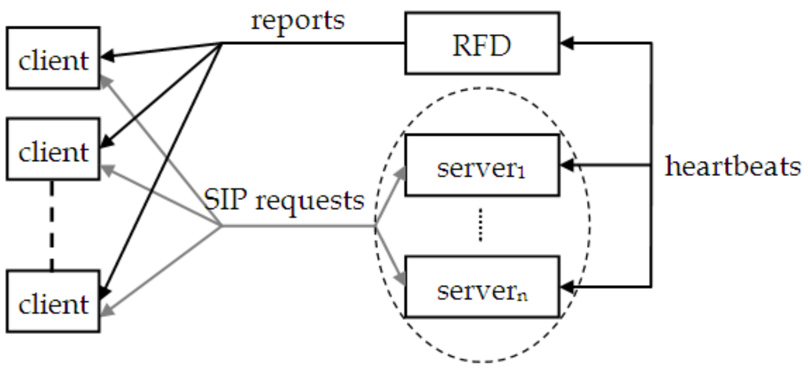


Figure 4.6: Network architecture of the replicated server application: clients send SIP requests to the server chosen based on the reports received from the RFD; the RFD refreshes reports based on heartbeats exchanged with the servers.

Service overview The client regularly requests the service to a given server by sending it a specific "SIP request" message; the choice of the server to contact is made by the client with the help of an additional entity, the Remote Failure Detector (RFD)[12]. The RFD is in charge of managing the registration (and de–registration) of the servers to the replicated server set, regularly checking the state of every server by heartbeats, and reporting the server set status to the clients by using specific messages named "SIP reports". The details about the strategies used to check servers, create reports, send reports and choose the server to contact are described in the following section 4.4.2. Figure 4.6 depicts the network architec-

10 RSerPool (Reliable Server Pooling) is an application–independent set of services and protocols for building fault–tolerant and highly available client/server applications.
11 SIP (Session Initiation Protocol) is an application-layer control (signaling) protocol for creating, modifying, and terminating sessions with one or more participants; these sessions include Internet telephone calls, multimedia distribution, and multimedia conferences.
12 The RFD is named "Name Server" in the Reliable Server Pooling architecture.

ture of the scenario considered, showing the clients, the servers, the RFD and the type of messages exchanged among nodes.

The above management of the redundant servers is an instance of private diagnosis and reconfiguration activities from the viewpoint of the client node (see the "private diagnosis" scenario described in section 2.2): the private diagnosis step aims to judge whether the monitored servers are available or not; the reconfiguration step consists in requesting retransmissions to the same server to cope with transient communication faults or in dropping the transaction with a faulty server (server failover) to pick–up another server and restart the request.

4.4.1 *Fault model and assumptions*

This section describes the fault model and the assumptions on which the model is based on. Some parameters are introduced in the following descriptions; the full list of parameters considered in the evaluation work is shown in table 4.9.

Server faults

Servers suffer crash faults; different servers suffer independent faults. When a server crashes, it is repaired and restored after some time; the server failure mode follows hence an exponentially distributed ON/OFF model, with mean time to failure *TTF* and mean time to repair *TTR*. The probability that a single server is ON or OFF can be easily computed from the *TTF* and *TTR* values; we refer to these probabilities as P_{OFF} and P_{ON} respectively.

We assume stateless SIP sessions, i.e. servers do not maintain session states and hence there cannot be inconsistencies (value faults are not considered). We assume perfect clock synchronization and thus we do not consider the potential effects of clock drift.

Remote Failure Detector faults

We assume that the RFD is not affected by internal faults; the RFD may have an inaccurate view of the state of the servers according to the communication faults (external faults).

Communication faults

Network errors (e.g. router buffer overflow, packet corruptions on wireless links) are always mapped into packet losses (e.g. via the use of CRC codes). We assume that heartbeats, reports, and SIP messages consists of only one packet each[13]: therefore, each packet loss is equivalent to the loss of a whole message. Packet losses occur with the same probability *PER* (Packet Error Rate) on both the uplink (from client to server) and the downlink (from server to client).

Communication delays are assumed to be exponentially distributed with mean value *Delay* for both the uplink and the downlink. Because of the communication delay distribution (whose variability is mimicking variations in network congestion levels), some heartbeats and SIP messages may transit in

13 Many SIP messages, such as `INVITE` and `BYE`, as well as typical chat messages do not exceed a few hundred bytes; heartbeats and report messages can be expected to be smaller than SIP requests/responses.

the network longer than their respective timeout allow; we assume that the responses to these messages are dropped because of timing failures.

The one–way delay and the packet error rate are independent for each pair of communicating endpoints. Malicious intrusions/message modifications are not considered.

Application traffic model

We assume each client sends a new SIP request some time after the previous transaction has been successfully completed or dropped (it is dropped when the maximum number of retransmissions is reached); this assumption implies that the client never manages multiple transactions in parallel. The time interval between the end of the previous transaction and the begin of next one, named inter–transaction time, follows an exponential distribution with mean value *InterSIP*.

Each message type is assumed to have a deterministic size: an heartbeat is 100 bytes, a report is 200 bytes, a SIP message is 400 bytes.

4.4.2 *Failure detection and reconfiguration strategies*

This section details the failure detection and reconfiguration strategies evaluated in the following sections. Failure detection is performed both in the RFD and in the client, whilst reconfiguration is performed in the client only.

Failure detection in the RFD

The RFD proactively checks servers by sending them heartbeats every *InterHB* seconds. The RFD then collects the check results in a SIP report by applying the "Maximum Availability Server Selection Policy", which was demonstrated in [Bozinovski 07] to be the server selection policy offering the highest dependability levels for a similar server replication architecture. The SIP report lists the servers ordered based on heartbeat response time: the server that replied last to the heartbeat request is in first place, the server that replied second to last is in second place, and so on (as a LIFO[14] queue). The RFD proactively sends the report to the clients as soon as the report has been created, that is after the heartbeat period has expired.

Failure detection and reconfiguration in the client

The client sends a SIP request to the first server listed in the last SIP report received from the RFD; if a successful response is received until a given timeout, the transaction is successful. If instead the timeout expires, a failure is detected and the client reacts by retransmitting again the SIP request to the same server. These retries last until either a successful response is received (successful transaction) or the maximum number *MaxRetrans* of unsuccessful retransmissions with the same server is reached; in this last case the client picks the next server listed in the SIP report and sends it a new request (failover of the old server). This mechanism is repeated until either the client receives a successful response, or the number *MaxFailOvers* of different servers contacted has been reached; in this latter case, the transaction is dropped. In the more likely case when the

14 Last In, First Out

transaction is successful, all counters are reset and the same server is used for the next transaction, unless a new SIP report is received during the inter–transaction time *InterSIP* and the selected server now appears to be unavailable.

The different instances of the reconfiguration strategies evaluated are obtained by changing the following (internal) parameters/characteristic:

- The timeout per request is managed into two different ways: exponential back–off vs. fixed timeout;
- The value for the *MaxRetrans* threshold;
- The value for the *MaxFailOvers* threshold;

Timeout per request: exponential back–off

The criteria behind managing the timeout per request as an exponential back–off follows the same criteria behind the timeout management of connection–oriented protocols at the fourth level of the ISO/OSI model (e.g. TCP): the timeout per request duration is increased by a multiplicative factor (here set to 2) for each subsequent retransmission. The initial timeout for a newly sent request is T_0. By default, the evaluation scenarios assume T_0 to be set to the average round trip time *RTT*. If the response has not been received within T_0, the client resends the request and doubles the timeout. In general the timeout value for the k^{th} retransmissions is given by $T_k = T_0 2^k$; the overall timeout since the first request is hence $Timeout = T_0(2^{k+1} - 1)$.

Timeout per request: fixed timeout

The alternative way of managing the timeout per request is to use the same fixed timeout for each retransmission. We set the timeout so that it does not expires on average for the 90% of the SIP requests/responses, which means setting it in such a way that the following equation holds (based on the assumption that the round trip time is exponentially distributed with mean value *RTT*):

$$p\left(RTT \leqslant T_{90\%}\right) = 1 - \exp\left(-\frac{1}{RTT}T_{90\%}\right) = 0.9$$

If RTT=100 ms then $T_{90\%}$=230 ms.

4.4.3 *Quantitative analysis*

This section presents the quantitative analysis of a few failure detection and reconfiguration strategies used to manage the replicated server pool. The relevant measures of interest are identified and the relevant parameters are described; the model representing the system is described and finally the results of the analysis performed are presented and discussed.

The quantitative analysis aims to understand the impact of the parameters configuring the failure detection and reconfiguration strategy on the measures of interest. The design choices that are going to be investigated are whether favoring retransmission or server failovers, whether using a fixed timeout or an exponential back–off, and how much the environmental parameters (e.g. server failures or communication faults) impact on those choices.

The system was modeled using the SAN[15] formalism [Meyer 85] and solved using the Möbius tool [Daly 00]. This choice was motivated by the flexibility and power found in SANs (and Möbius) in supporting a hierarchical modeling paradigm and a solution technique appropriate for the evaluation to be performed.

Different studies were performed on the modeled system varying some parameters; the relevant parameters are briefly explained hereafter.

Environmental parameters

The parameters related to the fault assumptions are the probability of server failures P_{OFF} and the probability of packet losses *PER*. They are both expected to impact on the effectiveness of the reconfiguration strategy, because retransmission are more effective when there are communication failures (*PER*), whilst failovers are more effective when servers are more likely to fail (P_{OFF}). The scenarios evaluated were selected setting the combinations of parameter values shown in table 4.7. The values shown in Table 4.7 are overstated with respect to many real systems; they were set overstated for the purpose of the quantitative analysis, aiming to amplify their effects on the measures of interest.

Table 4.7: Combinations of values assigned to the environmental parameters.

	Selected scenarios			
	L-L	L-H	H-L	H-H
PER	1%	1%	10%	10%
P_{OFF}	1%	10%	1%	10%

Controllable fault–tolerance parameters

The parameters related to the tuning of the fault–tolerance strategy are the *MaxRetrans* threshold, the *MaxFailOvers* threshold and the number of servers in the pool. Three different configurations of the standard reconfiguration strategy were selected; these configurations, detailed in table 4.8, represent two opposite configurations and a trade–off between the two opposites: "ReT" heavily

Table 4.8: Combinations of values assigned to the controllable fault–tolerance parameters.

	Reconfiguration strategies		
	ReT	Bal	FOvr
MaxFailOvers	1	3	6
MaxRetrans	3	1	0

15 Stochastic Activity Network

relies on retransmissions, "Bal" is a compromise between retransmissions and failovers, "FOvr" only uses failovers. The values listed in table 4.8 were chosen so that the maximum number of requests per transaction, named *MaxRequests*, is as close as possible to 7, which is the default value for the standard SIP setting. The formula linking *MaxRequests*, *MaxRetrans* and *MaxFailOvers* is the following:

$$MaxRequests = (MaxRetrans + 1) \cdot (MaxFailOvers + 1)$$

Each recovery configuration was evaluated for three different pool sizes. The basic size of the server pool for each configuration was set to *MaxFailOvers*+1; in addition to the basic size, two other sizes were considered: *MaxFailOvers*+3 and *MaxFailOvers*+5.

The quantitative analysis aimed to evaluate how these parameters impact on the measures of interest.

Measures of Interest

We are interested in measuring some metrics related to dependability and performance of the service, focusing mainly on the user perception of those service attributes.

Service availability (Dependability)

The relevant dependability attributes from the end–user perspective are service availability, content integrity and confidentiality. Based on the assumptions discussed in section 4.4.1, faults regarding content integrity and confidentiality are assumed out of scope, so that only server crash failures and message losses are considered. Service availability is hence defined based on the perception of a successfully provided service.

Service availability is defined as the ratio of successful transactions over the total number of transactions executed; this measure reflects on how well the fault–tolerance mechanism masks node and network faults.

Service access time (SAT)

From a performance perspective, the user is interested in perceiving the shortest transaction completion time. Node failures and network failures require request retransmissions, which in turn lead to longer transaction completion time. Transaction completion time can be minimized if clients ideally contact an available server every time they send a request; this depends also on the accuracy of the RFD, which is in charge of suspecting which servers are down and notifying this information to the clients.

Service Access Time (SAT) is defined as the average time between the time instant a transaction is started (i.e. when the request is sent for the first time) and the time instant the transaction is completed (i.e. when the client receives the response from the server). Only successful transactions are considered for the evaluation of SAT.

Induced load (Load)

Finally, one operator–centric metric is considered: the network traffic due to the specific strategy used to manage redundancy. It is important to measure how much network traffic is needed, because telecommunication systems in general, and wireless systems in particular, have limited bandwidth and perform worse with heavier loads, which in turn slows down the service execution and may even lead to service unavailability. Consequently, it is important to measure how much network traffic a specific fault–tolerance configuration produces.

The induced load is defined as the overall volume of end–to–end messages sent during an evaluation run, normalized to one transaction only. The normalization step is required because the application traffic model considered (see section 4.4.1) does not generate the same total number of transactions over a given simulated time interval.

Model

The Möbius framework allows the construction of composed models from previously defined models, which permits to adopt a hierarchical approach to modeling by creating sub–models as meaningful units and then placing them together to construct the model of a system. Model composition is accomplished by state–sharing, which links sub–models together; sub–models can also interact by both reading from and writing to a set of common state variables.

Replicate/join formalism

Möbius allows sub–models be composed by using the *replicate/join* operators: the *replicate* operator is used to construct a model consisting of a number of identical (indistinguishable) copies of its single child; the *join* operator is used to compose two or more sub–models using state–sharing. A special trick is used in replicated sub–models in order to give each replica a unique identity (see details in the description of the Server atomic model) and treat them independently.

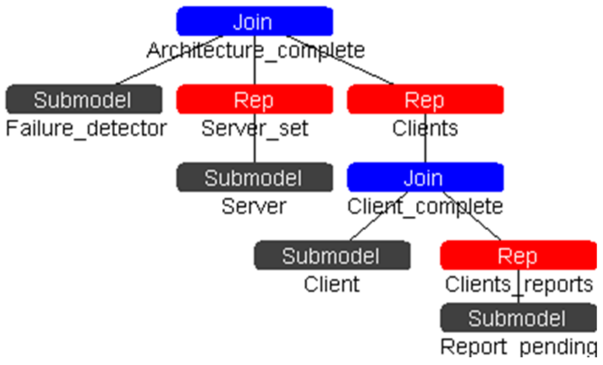


Figure 4.7: Composed model of the replicated server architecture: each leaf corresponds to a sub–model modeling a different system entity; internal nodes are used to join or replicate their sons.

The overall model

The translation of the network architecture shown in figure 4.6 into a SAN composed model is shown in figure 4.7: each leaf corresponds to a sub–model modeling a different system entity; internal nodes are used to join or replicate their children. In the following, we give a high–level description of the SAN sub–models of the composed composed model; all the details of the model are in [Lollini 08].

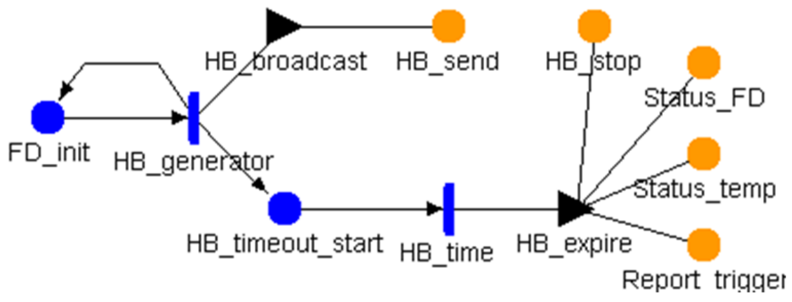


Figure 4.8: Atomic model of the remote failure detector.

RFD The RFD is modeled by the Failure_detector atomic model shown in figure 4.8. The RFD is in charge of implementing the heartbeat mechanism, managing the server status information and triggering the report messages; all these actions are modeled as described hereafter.

The RFD periodically broadcasts an heartbeat (timed activity *HB_generator*) to all servers (extended place *HB_send*, shared with the server models). When a heartbeat round is started (place *HB_timeout_start*), the RFD also starts the heartbeat timeout (timed activity *HB_time*). Upon expiration of the heartbeat timeout, the RFD performs the following actions:

1. Discards any pending heartbeat with servers from which a response has not been received (extended place *HB_stop*, shared with the server model);
2. Updates its cached status information (extended place *Status_FD*) with the information gathered during the heartbeat round (extended place *Status_temp*), and resets the latter;
3. Creates a new report message and sends it to all clients (extended place *Report_trigger*, shared with the client model).

Server The server is modeled by the Server atomic model shown in figure 4.9. The top–left subnet shown in the figure allocates a unique ID number (stored as the marking of place *ID_server*) to each server at the beginning of the evaluation run. The top–right subnet shown in the figure models the failure model of the server. At start up the server is ON: this information is stored in the *UP* extended place, in particular in the entry corresponding to the ID of the specific server. The timed activity *TTF* (exponentially distributed with rate *TTF*) models the server failure (*DOWN*), whilst the timed activity *TTR* (exponentially distributed with rate *TTR*) models the server repair.

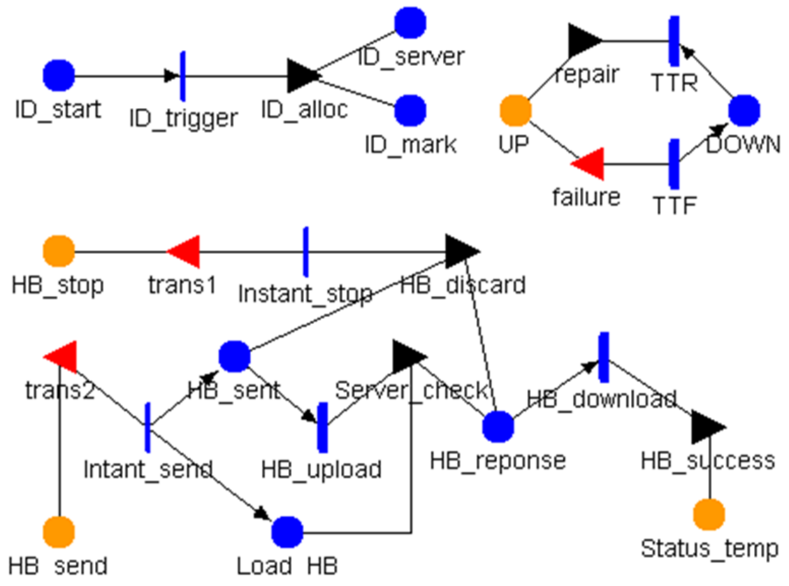


Figure 4.9: Atomic model of a server.

The rest of the net shown in figure 4.9 models the independent heartbeat communications between the RFD and the server. Once the heartbeat is sent by the RFD (firing of the timed activity *HB_upload*), it can either be lost because of a communication fault, or reach the server, giving the chance to check the server status (the output gate *Server_check* reads directly the proper entry of the *UP* extended place). If the server is ON, a response is sent back to the RFD (place *HB_response*); if not, the subnet stays idle until the current heartbeat is discarded. If the response is not lost on the downlink, the status information is updated at RFD (extended place *Status_temp*, shared with the RFD model). In case the heartbeat timeout expires before the heartbeat sequence is complete, a marking change in *HB_stop* triggers the *Instant_stop* activity, which in turn executes the *HB_discard* output gate, preventing stale pending heartbeats from updating the status information illegally.

Client The client is modeled by the Client atomic model shown in figure 4.10. The top subnet shown in the figure allocates a unique ID number (stored as the marking of place *ID_client*) to each client at the beginning of the evaluation run. The subnet beneath the top one allocates a dedicated sequence number and instances a Report_pending sub–model to each new report message generated and sent by RFD. This mechanism implements multiple concurrent instances of the communications between RFD and the clients. This is necessary because, as opposed to heartbeats and SIP requests, there is no timeout for the report messages: a client does not discard an incoming old report if this report is at least more recent than the report that the client is currently caching.

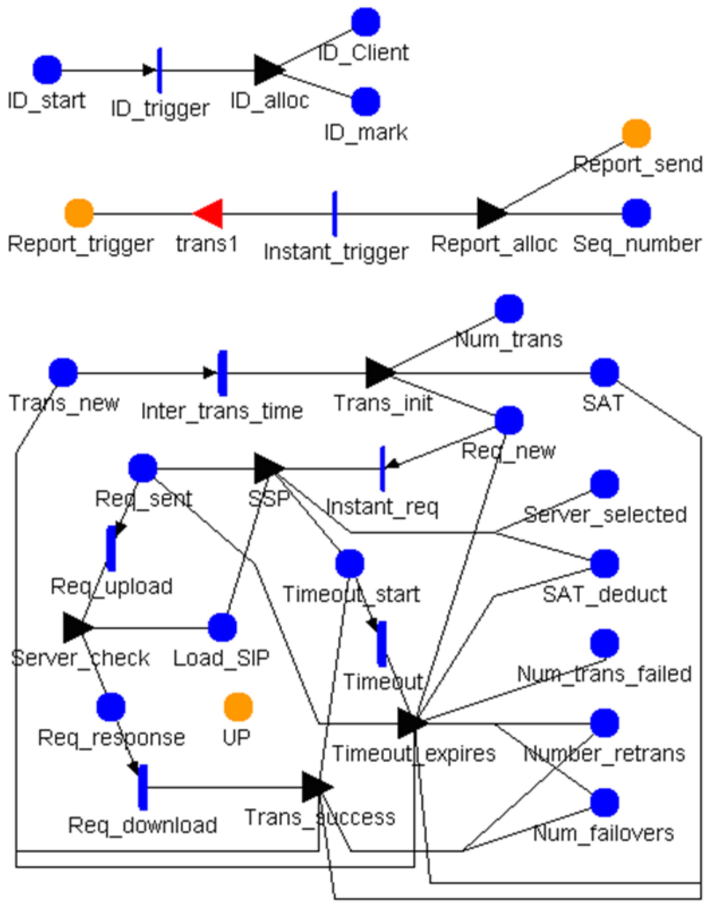


Figure 4.10: Atomic model of a client.

The bottom subnet shown in figure 4.10 models the SIP traffic generator and the associated failure detection and recovery schemes. When a new transaction is started (firing of the timed activity *Inter_trans_time*), the SIP request is prepared (*Req_new*) by selecting a server ID (the chosen ID is saved as the marking of place *Server_selected*) according to the server selection policy (implemented in output gate *SSP*). Then, the request is sent and the SIP timeout–per–request started (timed activity *Timeout*). If the request is not lost on the uplink, the status of the selected server is checked (the *UP* extended place is shared with the Server sub–model), and the response is sent back toward the client. In case the transaction is successful (output gate *Trans_success*), the client is ready for the next transaction to start (*Trans_new*); in this case the following actions are performed: the timeout is stopped by emptying the place *Timeout_start*, and the retransmission and failover counters (modeled by places *Number_retrans* and *Num_failovers* respectively) are reset to zero. Furthermore, the same server is maintained by the server selection policy for the next transaction, unless during the time of activity *Inter_trans_time* a new SIP report is received indicating that the server is no longer available.

If instead the timeout expires before the transaction has completed, the pending request is discarded, the retransmission counter (place *Number_retrans*) is incremented, and a new request is generated (*Req_new*). When the counter of the retransmissions (modeled as the marking of *Num_retrans*) reaches the *Max_Retrans* threshold, a failover is triggered, and the following actions are performed: place *Num_retrans* is emptied, the marking of *Num_failovers* is incremented, the place *Server_selected* is emptied (so that SSP selects another server), and a new request is generated.

When both the recovery counters - which are modeled by places *Num_retrans* and *Num_failovers* - reach their thresholds (*Max_Retrans* and *Max_FailOvers* respectively), the current transaction is discarded and a new one is prepared by resetting the recovery counters and the selected server ID to zero.

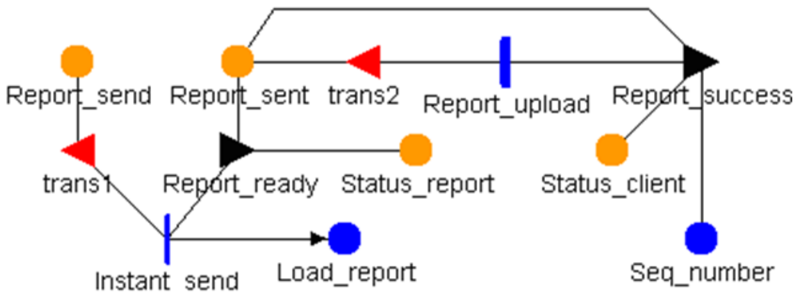


Figure 4.11: Atomic model of a SIP pending report.

SIP report The SIP report is modeled by the Report pending atomic model shown in figure 4.11. This model models the behavior of communication between the RFD and the specific client instance with which the current report is joined to in the composed model. Before uploading the report to the client, the RFD includes the server set status in the message (modeled in the extended place *Status_report*, shared with the RFD model). When a report is successfully received at the client side, the client updates its cached report (extended place *Status_client*) and discards any pending report with a lower sequence number than the last one received (action performed by the output gate *Report_success*).

Reward structures The evaluation of the measures of interest in Möbius involves specifying reward variables and defining a reward structure for each reward variable.

Service availability: every time a new transaction is started, the marking of place *Num_trans* is incremented; analogously, when a transaction fails the marking of *Num_trans_failed* is incremented. Service availability is hence evaluated as:

$$\text{Service availability} = 1 - \frac{\mathit{Num\_trans\_failed} \rightarrow \text{Mark()}}{\mathit{Num\_trans} \rightarrow \text{Mark()}}$$

Average response time: the marking of places *SAT* and *SAT_deduct* is used to evaluate the overall time spanned by transactions over the simulation run. The average transaction time is hence evaluated as:

$$\text{SAT} = \frac{\texttt{Time}(SAT) - \texttt{Time}(SAT\_deduct)}{Num\_trans \rightarrow \text{Mark}() + Num\_trans\_failed \rightarrow \text{Mark}()}$$

Load induced: each sub–model replica increments its respective load counter (*Load_HB*, *Load_report*, *Load_SIP*) when it sends a message. The load per transaction is hence evaluated as:

$$\text{Load} = \frac{Load\_HB \rightarrow \text{Mark}() + Load\_report \rightarrow \text{Mark}() + Load\_SIP \rightarrow \text{Mark}()}{Num\_trans \rightarrow \text{Mark}()}$$

Model evaluation and System analysis

This section presents the results of the evaluation of the measures of interest performed on the failure detection and reconfiguration strategies presented above.

Table 4.9: Model parameters, their default values and their description.

Name	Default Value	Meaning
TTR	10	Mean value for the exponentially distributed Time To Repair (sec)
P_{OFF}	0.01	Probability of a server being failed (DOWN)
CL	1000	Cycle Length (CL=TTF+TTR): mean interval between crashes at a single server (sec)
InterHB	5	Time interval between heartbeats sent by the RFD to check the servers (sec)
PER	1	Packet Error Rate (%)
Delay	100	Mean value for the exponentially distributed communication delay (ms)
RTT	100	Mean value for the exponentially distributed Round Trip Time (ms)
T_0	*RTT*	timeout for the first SIP request (ms)
MaxRetrans	1	Max number of unsuccessful request retransmissions with the same server
MaxFailOvers	3	Max number of failovers to drop a transaction
Clients	10	Number of clients in the simulation
InterSIP	5	Mean value for the exponentially distributed inter-transaction time (sec)

The measures of interest were evaluated by using the simulation solver instead of a numerical solver; this choice was motivated by the large state–space description of the model. Each test run was set to last 20 hours (i.e. 72000 seconds) of simulated operation time. All setting scenarios evaluated were run a minimum of 6 times, converging within 95% probability in a 0.1 relative interval. Table 4.9 shows the relevant model parameters and their default values.

Study at varying recovery configuration and pool size

The first study analyzes three different recovery configurations of the standard replicated SIP strategy, all three managing the timeout per request as an exponential back–off with basic timeout per request set to T_0=*RTT* (i.e. 100 ms). The three recovery configurations analyzed are "ReT", "Bal" and "FOvr"; they differ because of the values assigned to the maximum number of retries *MaxRetrans* and the maximum number of failovers *MaxFailOvers*, as shown in table 4.8. A fourth recovery configuration is considered in order to have a reference value: the standard SIP strategy without replication, that is a dummy strategy encompassing only 1 server in the pool. Each recovery configuration is analyzed at varying the number of servers in the pool and for 4 different scenarios, each corresponding to different settings of the environmental parameters *PER* and P_{OFF} (the details are in table 4.7): "L-L", "L-H", "H-L" and "H-H".

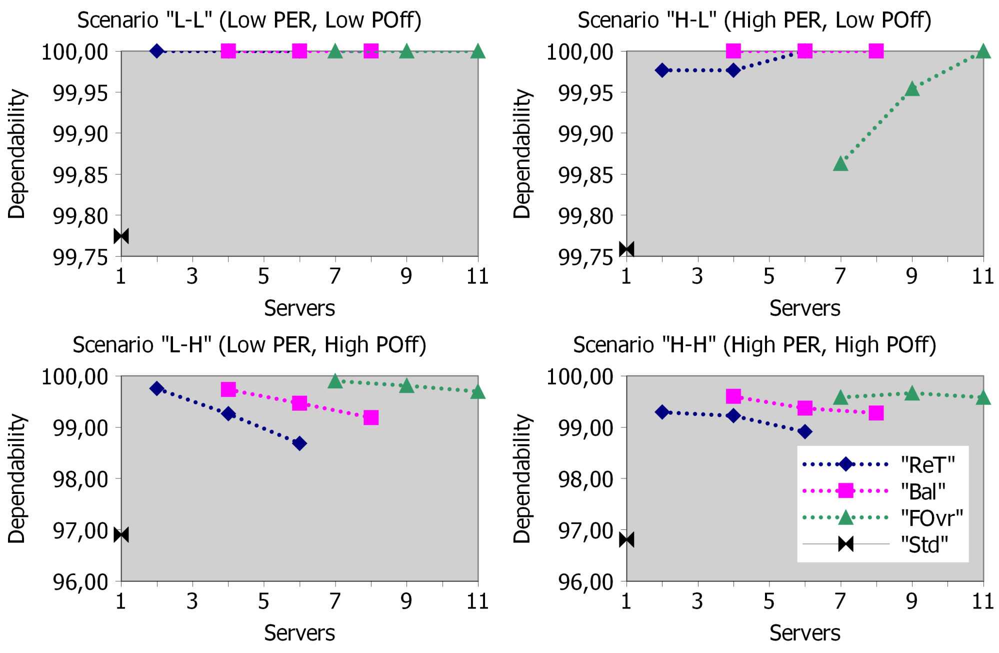


Figure 4.12: Dependability at varying the number of servers in the pool; three configurations of the SIP strategy (plus the non replicated one) are analyzed in 4 different scenarios each.

Figure 4.12 shows how dependability changes at varying the number of servers in the pool: the figure is divided into four sub–figures, one for each scenario considered. Each sub–figure shows three curves, one for each recovery configuration evaluated, and a black marker, showing the result obtained by the standard non replicated SIP strategy. Figures 4.13 and 4.14 have the same structure as figure 4.12, but show SAT and Load respectively.

Dependability

Figure 4.12 shows that in general dependability increases when the number of servers in the pool increases, with the exception of the "FOvr" recovery strategy in the "H-L" scenario, where the more servers there are, the lower is the resulting dependability. This is due to the combined effect of many servers failures (P_{OFF} is 10%) and a recovery strategy using failovers only ($MaxRetrans = 0$). The recovery strategy showing the best compromise across all the scenarios considered is "Bal". All the recovery strategies in all scenarios show dependability values higher than the reference value corresponding to the non replicated strategy, which confirms the intuitive reasoning that is better to have replication in any case, no matter which is the scenario or the particular reconfiguration strategy in place.

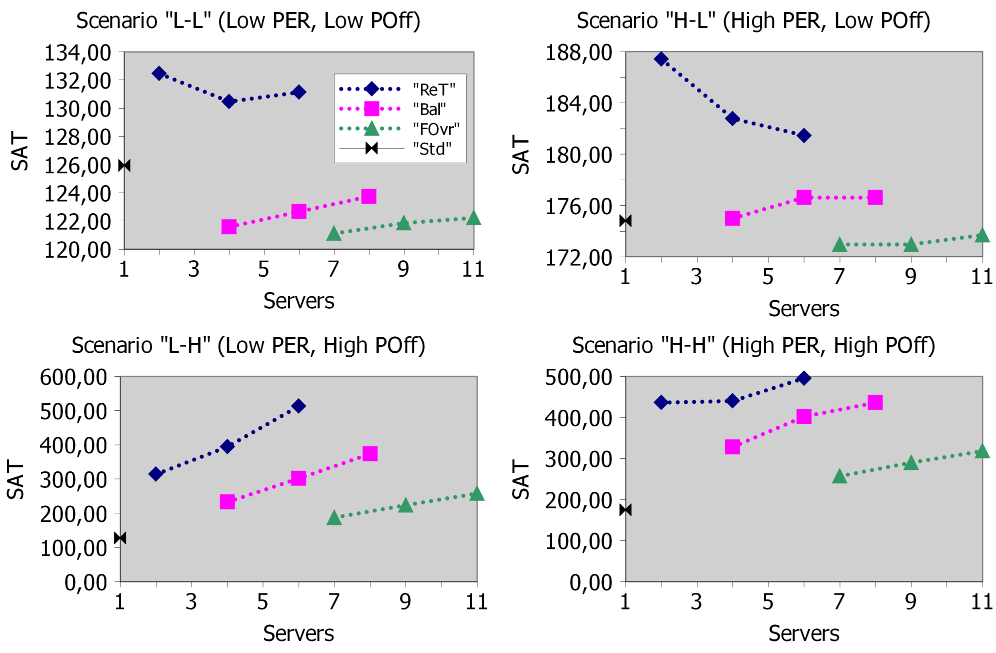


Figure 4.13: Service Access Time (SAT) at varying the number of servers in the pool; three configurations of the SIP strategy (plus the non replicated one) are analyzed in 4 different scenarios each.

SAT

Figure 4.13 (page 135) shows that in general an increasing number of servers leads to higher service access time; with the exception of the "ReT" strategy in the "H-L" scenario. This is due to the combined effect of a high *PER* (here set to 10%) and a strategy recovering mostly via retransmissions. The strategy showing in general the lower values for SAT is "FOvr", the one showing the higher values is "ReT". The reference value is lower than all the reconfiguration strategies in the "L-H" and "H-H" scenarios, whilst it is larger than "Bal" and "FOvr" in the "L-L" scenario and it is larger than "FOvr" in the "H-L" scenario. This is due to the fact that the more retransmissions are sent to the same server, the longer is the timeout and hence the service access time.

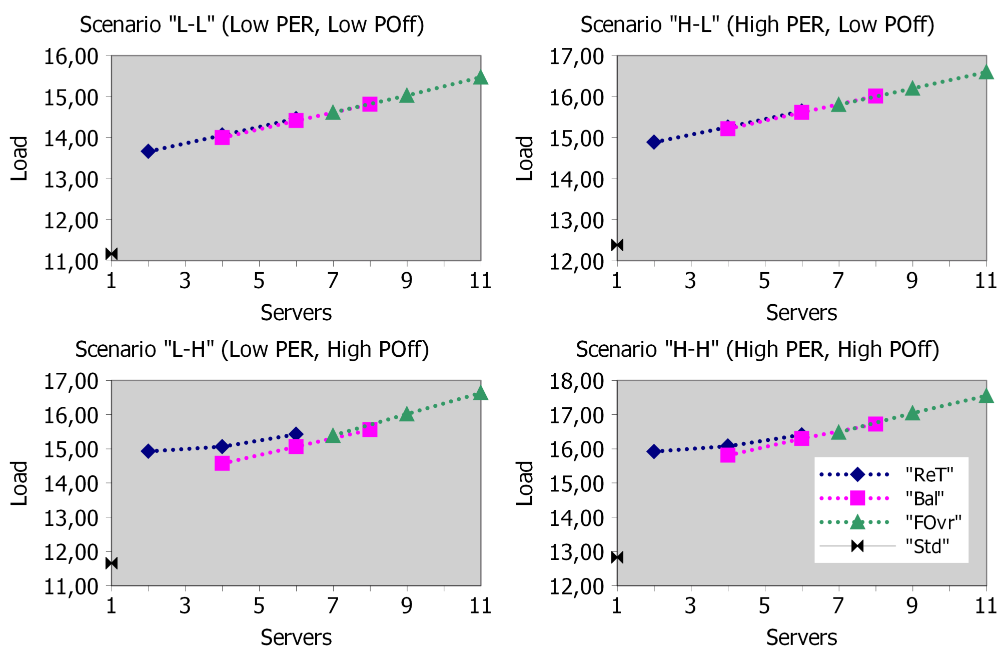


Figure 4.14: Load at varying the number of servers in the pool; three configurations of the SIP strategy (plus the non replicated one) are analyzed in 4 different scenarios each.

Load Figure 4.14 (page 136) shows that in general an increasing number of servers leads to a linear increment in the Load for all the recovery strategies considered in all scenarios. All the recovery strategies considered show a larger Load than the standard strategy; this is true in all scenarios considered.

Study at varying the timeout–per–request The second study analyzes the three reconfiguration strategies "ReT", "Bal" and "FOvr" at varying the management of failure detection at the client side. Two different mechanisms for managing the timeout–per–request are considered: i) the standard exponential back–off ("EXP"), and ii) the fixed timeout ("FIX"). The details of both mechanisms are in section 4.4.2. The standard non replicated SIP strategy is also considered and used as reference value (this strategy uses the exponential back–off). Each recovery configuration was analyzed in 4 different scenarios (the same as those of the first study), each corresponding to different settings of the environmental parameters *PER* and P_{OFF} (the details are in Table 4.7): "L-L", "L-H", "H-L" and "H-H".

Dependability Figure 4.15 shows the values for dependability for both "EXP" and "FIX".

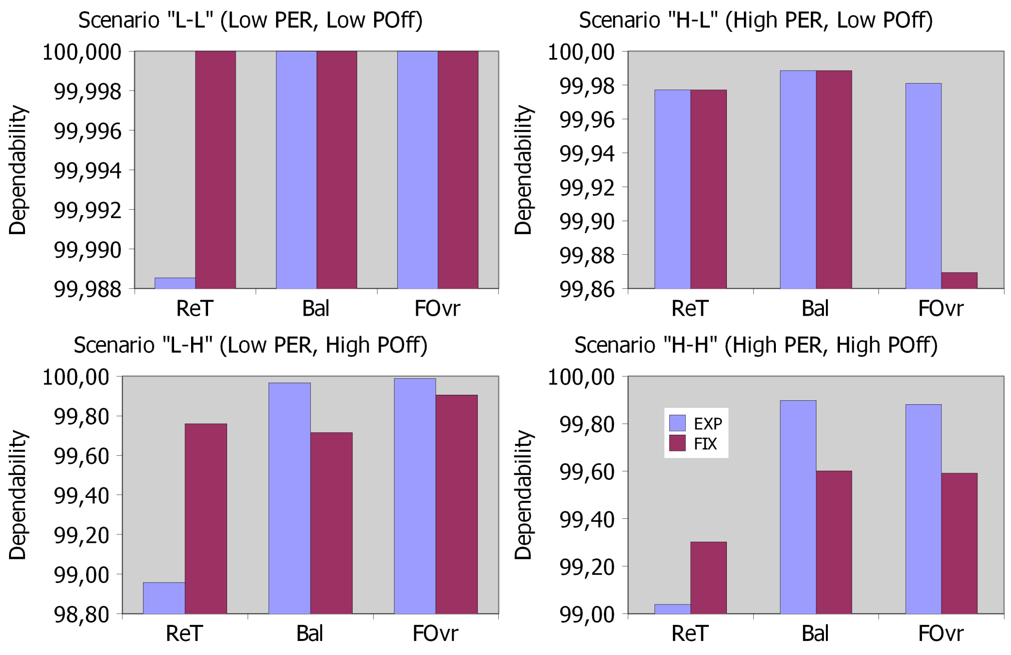


Figure 4.15: Dependability at varying the failure detection strategy at client side (exp. back–off vs. fixed timeout); 3 configurations of the SIP strategy are analyzed in 4 different scenarios each.

Comparing the values obtained by "EXP" with those obtained in the same settings by "FIX" it emerges that "EXP" has lower dependability when more retransmission are used ("ReT") and instead it has higher (sometimes the same) dependability when the other two strategies are used ("Bal" and "FOvr"). This trend is particularly evident for high P_{OFF}, when the number of failover increases.

The fixed timeout setting does not help in scenarios favoring retransmissions, because the transaction lifetime per server becomes shorter, lowering the probability that a server has recovered by the time the last retransmission is sent to that server. Given the parameter values considered, i.e. T_0=100 ms and $T_{90\%}$=230 ms, the transaction lifetime per server is almost the same for the second retransmission and becomes much shorter from the third retransmission in the fixed timeout case (920 ms against 1500 ms). When the number of retransmissions is lower than 3, as in the case of "Bal" and "FOvr", the transaction lifetime per server is actually larger with $T_{90\%}$, and hence dependability increases.

SAT Figure 4.16 (page 138) shows the values for service access time. Comparing the values obtained by "EXP" with those obtained in the same settings by "FIX" it emerges that "EXP" has lower SAT in all the scenarios considered. This is the positive effect of avoiding some of the retransmissions caused by long com-

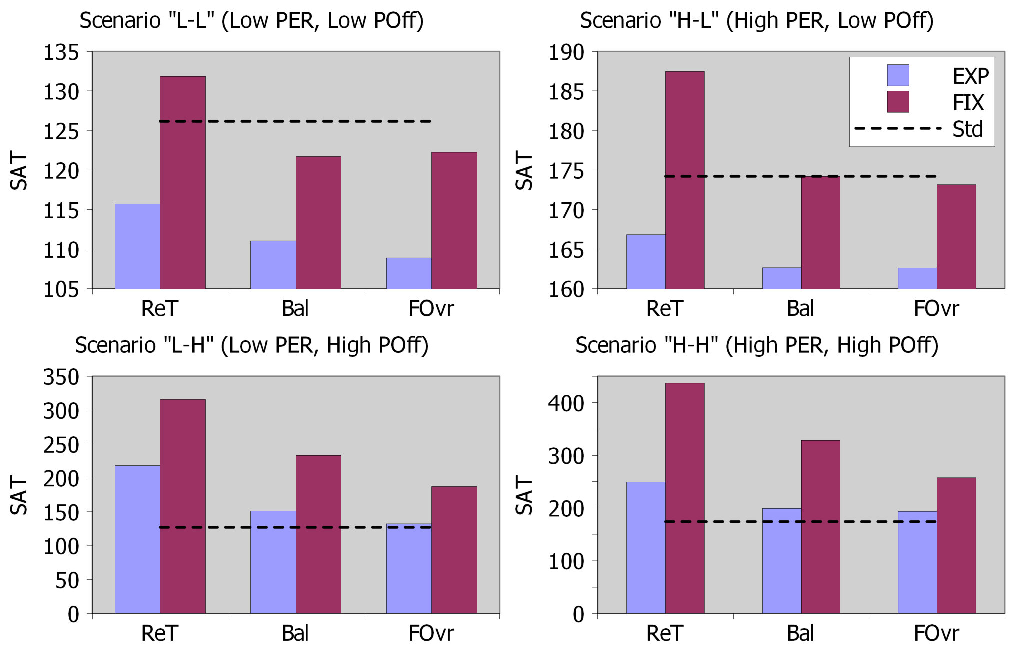


Figure 4.16: Service Access Time (SAT) at varying the failure detection strategy at client side (exp. back–off vs. fixed timeout); 3 configurations of the SIP strategy (plus the non replicated one) are analyzed in 4 different scenarios each.

munication delays. "EXP" shows also lower SAT than the non–replicated SIP strategy in those scenarios where P_{OFF} is low ("L-L", "H-L").

Load Figure 4.17 (page 139) shows the values for Load. Comparing the values obtained by "EXP" with those obtained in the same settings by "FIX" it emerges that "EXP" has lower Load in all the scenarios considered. Both "EXP" and "FIX" show higher Load than the non–replicated SIP strategy in all scenarios considered.

4.4.4 *Concluding remarks*

Evaluations performed has demonstrated that, for the given network characteristics and server fault model, it is in general preferable to favor failovers instead of retransmissions. Such a strategy has lead to higher dependability levels in almost all fault scenarios (except in the "H-L" scenario); moreover, the advantage of using many failovers instead of retransmissions toward the same server has been that the timeout value always stays low despite retransmissions (the timeout–per–request increases exponentially in this setting), so SAT has became shorter as well. This behavior has been even emphasized when the number of servers in the pool is incremented. The only drawback of this reconfiguration strategy has been that it has the higher Load (the strategy with lower Load is

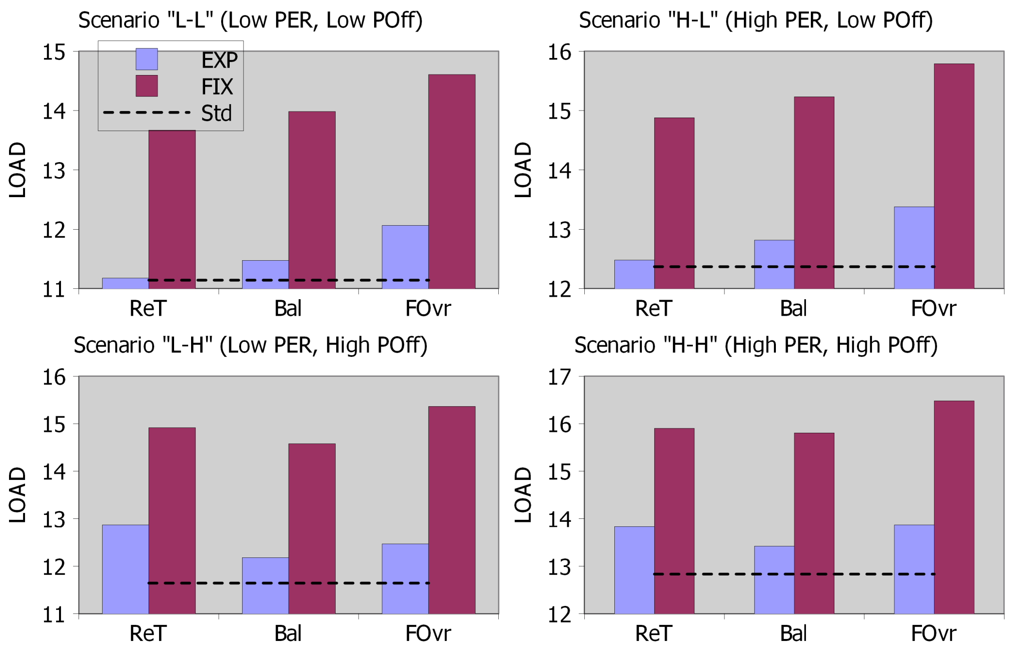


Figure 4.17: Load at varying the failure detection strategy at client side (exp. back–off vs. fixed timeout); 3 configurations of the SIP strategy (plus the non replicated one) are analyzed in 4 different scenarios each.

"ReT"). If instead Load has to be lowered down, it is in general preferable to favor retransmissions, at the cost of decreasing dependability and increasing service access time.

5 SECURITY IN CRITICAL INFRASTRUCTURES

This chapter describes an ongoing work for enhancing the security in SCADA[1] infrastructures; this work is performed in the context of the INSPIRE project[2] and focuses on security at communication layer. A SCADA system is a common process–automation system used to gather data from sensors and instruments located at remote sites, and to transmit and use this data at a central site for both monitoring and control purposes. This work aims to enhance the reliability of communications over unreliable and/or insecure links.

The design principles of a diagnostic mechanism for the detection of sinkhole attacks to the routing protocol of a wireless sensor network in a SCADA island are presented. This diagnostic mechanism is an instance of the private diagnosis proposed in section 2.2; the diagnostic mechanism works on–line, fed by monitoring entities, and it is implemented using a probabilistic approach.

Since the reported work is still ongoing, the following steps need to be completed at the time of writing: the selection of the relevant indicators feeding the diagnostic mechanism, and the tuning of the internal parameters of the diagnostic mechanism itself.

5.1 INSPIRE ARCHITECTURE OVERVIEW

This section describes in summary the architecture of the SCADA–based infrastructure considered in this work.

Since a SCADA system is an industrial measurement and control system, it is composed by the following components:

SENSORS AND ACTUATORS: sensors measure the status of specific system parameters; actuators are used to control the industrial process.

REMOTE TERMINAL UNITS: these components, also called RTUs, convert sensor signals to digital data, and send this data to the Supervisory Station.

SUPERVISORY STATION: (also called *Control Room* or *Operation Control Center*) it takes decisions about the commands that are going to be executed by the actuators, based on the data gathered from the RTUs.

COMMUNICATION INFRASTRUCTURE: this is the mean through which data and commands are exchanged among the above components.

1 Supervisory Control And Data Acquisition

2 INSPIRE is an ongoing project funded by the FP7 programme of the European Commission (EC grant agreement n. 225553). `http://www.inspire-strep.eu/`

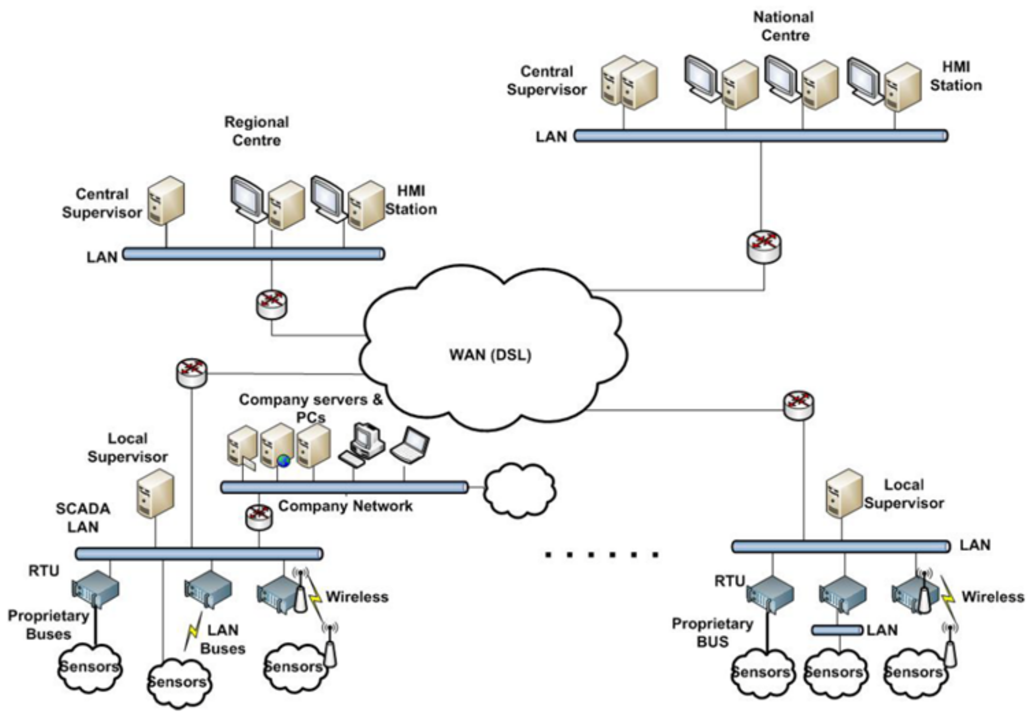


Figure 5.1: The architecture of a SCADA system: several islands (at different logical levels) are connected by a WAN; each island has a local supervisor and some RTUs managing sensors.

Figure 5.1 shows the architecture of a SCADA system encompassing the components listed above: it is evident the partition of the system in several SCADA islands interconnected by a WAN[3]. Each SCADA island has a local supervisor, managing the local sensors and actuators through its RTUs, which is in turn supervised by a regional/national supervisor (supervisors are logically organized in a tree structure).

Data flowing through the communication infrastructure is related both to the control of the industrial process (e.g. values collected by sensors, commands to be executed by actuators) and to the monitoring and reconfiguration actions of the SCADA system itself (e.g. alerts, keep–alive messages, reconfiguration requests). Data flows in the following directions: industrial information and monitoring information flows upstream, from sensors to supervisors, commands and reconfiguration requests flow downstream, from supervisors to actuator.

This work is focused within a SCADA island, in particular within the emerging wireless networks connecting sensors with RTUs.

3 Wide Area Network

5.2 PROBABILISTIC DIAGNOSIS OF SINKHOLE ATTACKS TO THE CTP PROTOCOL

This section presents the on–line diagnostic mechanism, based on HMM[4] (see section 1.2.2), for the diagnosis of sinkhole attacks in the wireless sensor networks (WSNs[5]) implemented in the SCADA islands. After the description of the scenario in which diagnosis has to be performed, the fault model and the assumptions are presented. Then the design principles for the monitoring and diagnosis activities are presented, together with the plan of the future activities devoted to complete this work.

5.2.1 *Scenario description*

The scenario considered in this work [Daidone 09] encompasses some wireless sensors that measure some physical parameters of the ambient in which they are spread; the measured values are then sent through a wireless connection toward the RTU.

This scenario comprises the following components:

- One RTU/local supervisor, which is a legacy server;
- One base station for a wireless sensor network[6], connected to the RTU through an USB connection;
- Some wireless sensors, connected to the base station through the wireless network; wireless nodes[7] are equipped with a sensor[8] measuring light and temperature.

Wireless nodes execute the TinyOS operating system[9] and use the CTP[10] protocol [Fonseca 07] as the routing protocol for forwarding the collected measures from sensors to the RTU.

The CTP routing protocol

CTP is an open–source best–effort routing protocol; it uses a shortest path first algorithm, giving priority to routes with the lower cost to the root node. Since CTP is a tree–based collection protocol, the base station advertises itself as the tree root, so that the wireless nodes form a routing tree to the base station. Routes are generated by using a routing gradient, implemented as a numeric

4 Hidden Markov Model
5 Wireless Sensor Networks
6 The base station is a CrossBow MIB520; `http://www.xbow.com/Products/productdetails.aspx?sid=227`.
7 The node is an Iris mote; `http://www.xbow.com/Products/productdetails.aspx?sid=264`.
8 The sensor is a CrossBow MDA100; `http://www.xbow.com/Products/productdetails.aspx?sid=178`.
9 TinyOS is an open-source operating system designed for wireless embedded sensor networks. `http://www.tinyos.net/`
10 Control Tree Protocol

value called ETX, which is associated with each node. Each node chooses the neighbor which has the lower ETX value (root node has ETX=0) as its parent node; it then updates its own ETX value, which is defined as the ETX of its parent plus the ETX of the link to its parent, and then advertises its ETX value to its neighbors. The ETX value for the communication link between two nodes is evaluated based on some communication–related values that each node periodically sends to its neighbors (e.g. LQI and RSSI values provided by the radio of the node[11], number of packets successfully received, number of packet transmitted).

Parent selection policy

A node changes its parent when one of the following conditions holds:

- There is at least a neighbor which declares an ETX value lower than a given threshold; the neighbor declaring the lowest ETX value is chosen as the new parent.
- The current parent node is congested and there is at least a neighbor node which declares an ETX value lower than a given threshold; the neighbor declaring the lowest ETX value is chosen as the new parent.

CTP packet types

Packets routed by the CTP protocol are classified in the following categories:

DATA PACKET: it contains the measures collected by the sensors (light and temperature);

ROUTING PACKET: it contains routing information used by nodes to update the values of their routing parameters;

CONTROL PACKET: it contains information about the current status of the sender node in terms of communication and routing parameters[12].

All the CTP packets are generated by the wireless nodes; data and control packets are forwarded until they are received by the root node, whilst routing packets are single-hop packets sent by a node to its neighbors.

5.2.2 *Fault model and assumptions*

The routing protocol suffers the *sinkhole attack*; the sinkhole attack is launched by a malicious node, the attacker, in order to exploit the routing and force the other nodes to use the attacker as their parent (or ancestor) in the route toward the root. Figure 5.2 shows an example of a routing tree before (left) and after (right) node 4 has successfully launched a sinkhole attack; arrows represent the

11 LQI (Link Quality Indication) is a characterization of the strength and/or quality of a received packet; RSSI (Received Signal Strength Indication) is a measurement of the power present in a received radio signal.

12 A control packet contains information about the sender node such as its ETX value, the number of neighbors, the number of packet sent, the number of packet dropped, etc.

parent–son relationship, with the arrow directed to the parent. The effect of the sinkhole attack is to direct the traffic toward the root through the attacker node.

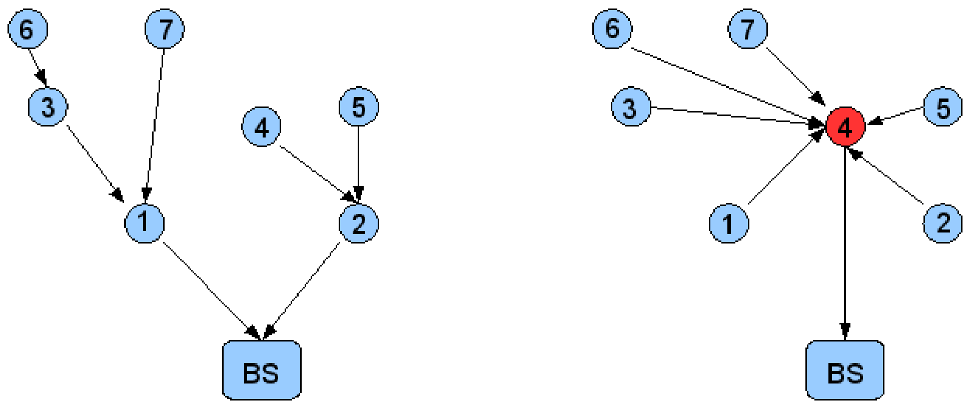


Figure 5.2: An example of a routing tree before (left) and after (right) node 4 has successfully launched a sinkhole attack; arrows represent the parent–son relationship, with the arrow directed toward the parent.

A successful sinkhole attack gives the attacker the opportunity to compromise the confidentiality of data, the integrity of messages and the availability of sensors. This is true especially in our scenario, since the CTP protocol does not implement any security defense. This is a design choice due to the fact that both traffic authentication and encryption require high CPU work, which in turn can quickly decrease the lifetime of battery–powered sensors.

When an attacker launches the sinkhole attack, it is assumed to perform the following actions:

- It sends routing packets with a rate higher than usual;
- It forges fake routing packets in order to advertise itself as an attractive parent;
- It forges data packets so that they report a low ETX value, advertising itself to be an attractive parent.

The routing protocol suffers the following problems:

LOOPS: there is a loop in the routing tree when a node selects one of its descendants as a new parent. A loop is detected when a node receives a data packet containing an ETX value lower than its ETX value. A loop is eliminated by forcing a refresh of the routing, which in turn corresponds to an increment of the routing packet rate of the nodes involved in the loop.

DUPLICATE DATA PACKETS: when a node receives a data packet which has to be routed toward the root, it send an `ACK` to the sender; if the sender does not receive this `ACK`, it retransmits the data packet. The problem of duplicate packets is solved thanks to a THL[13] field in the packet, which is incremented on each hop; a link–level retransmission has the same THL value, while a looped version of the packet is unlikely to do so.

It is assumed that there is at most one attacker in the network, and that the network cannot be partitioned. Moreover the topology of the network is assumed to be nearly static.

5.2.3 *The diagnostic mechanism*

The diagnostic mechanism is implemented in the RTU[14], aiming to diagnose whether one of the wireless nodes attached to the base station is an attacker or not; this is an instance of the private diagnosis scenario proposed in section 2.2, where the diagnostic mechanism diagnoses a remote entity based on its perception of the status of the remote entity. The RTU receives all data packets and control packets generated by the wireless nodes and uses the information reported within them in order to monitor the network and diagnose an attack.

Design principles

The design choice of implementing a centralized diagnostic mechanism has some advantages and drawbacks when compared to a distributed approach. Advantages are the following: i) the diagnostic mechanism runs on top a machine and hence it can be extremely complex[15], for example to search for complex correlations; ii) the consistency and control of the state of the SCADA subsystem is easier to reach. Drawbacks are the following: i) the detection latency is higher; ii) the centralized mechanism can be unable to detect attacks targeting the routing layer when the attacker is able to prevent packets bringing relevant information for the diagnostic mechanism to reach the RTU. All these drawbacks will be mitigated in INSPIRE by using the hybrid detection approach sketched in [Daidone 09]: each node runs a simple *local detection agent* which is in charge of identifying suspicious nodes; suspected nodes are not used for routing and are temporarily inserted in a black list until a judgment comes from the centralized diagnostic mechanism. The suspected node can be either canceled from the black list, not being an attacker, or permanently inserted in the black list, being an attacker. The integration between the centralized diagnostic mechanism and the local detection agents implemented in the wireless nodes is still an ongoing work and it is not described here.

Implementation outline

The diagnostic mechanism is implemented by using the probabilistic ap-

13 Time Has Lived
14 Remote Terminal Unit
15 The machine is assumed to have enough resources (e.g. in terms of memory, computational power, power supply) to support the mechanism.

proach based on HMM presented in 1.2.2. An HMM is built for each monitored node, defining the model states as follows: the node can either be an attacker or not, so at least two states can be identified, namely *Healthy* and *Attacker*. If the monitored node is not an attacker, it can be either a victim of an attack or not, so it seems effective to define a third healthy state: *Victim*.

Monitored events

The definition of the symbols observed by the HMM derive from which indicators are going to be monitored over time; since the selection of the relevant indicators is still an ongoing work, an overview of possible meaningful indicators is given hereafter. The following events can be monitored in the RTU for each wireless node and used as symptoms:

1. Node i increments the number of routing packets transmitted; this event can be a symptom that node i is an attacker, but it can also happen due to normal activity: the routing needs to be updated due to communication problems or node i received requests for routing information from a new neighbor.

2. An increment in the number of routing packets received by node v; this event can be the a symptom that node v is victim of an attack, but it can also be due to normal activity: e.g. a loop involving node v is going to be detected/broken.

3. The ratio between the number of routing packet sent and received for node v has an anomalously low value: this can be the symptom that node v is a victim, because the attacker sends a higher number or routing packets than usual, whilst the other nodes, keeping their normal behavior, send a low number of routing packets.

4. Node v discovers a new neighbor.

The WSN topology, inferred from the packets received over time by the RTU, is used to correlate over time the diagnosis of each wireless node and detect possible sinkhole attack scenarios. The definition of the correlation rules is still coarse and need to be tuned. Here are some general rules:

- Some nodes change their parent during a given time window and all (or almost all) choose node i as their parent: this can be the trace that node i is an attacker (and the others are victims), but it can also be the effect of a not malicious routing update.

- Some nodes, diagnosed as victims, share the same parent: the parent is likely to be an attacker.

5.2.4 *Future work*

This section gives a high–level view of the activities that need still to be completed. Some of the uncompleted activities need to be adjusted based on practice, but we still lack the feedback from the practice (e.g. the tuning of the HMMs).

The activities that need still to be completed are the following:

1. The identification of the relevant indicators for ongoing attacks; the plan is to select the relevant indicators by starting from a large set and then refining it by using the learning algorithm for HMM presented in section 2.4.

2. The tuning of the diagnostic mechanism for the single node; this step will be performed by using the learning algorithm.

3. The definition of the correlation rules used to infer attack scenarios.

4. Evaluate some dependability and/or performances measures at varying some internal parameters in order to analyze the trade–off between completeness and accuracy.

6 CONCLUSIONS

This thesis has described the work done in the last three years on diagnosis in critical infrastructures.

The work has been motivated by the observation that critical infrastructures are changing in the last decade. They were originally developed from proprietary architectures, where ad hoc solutions were chosen and several components were developed independently. Due to technological advances, deregulations and market liberalizations, critical infrastructures are evolving by progressively becoming larger in scale and more complex. These infrastructures are growing by incorporating unstable COTS components, old legacy machinery or previously disjoint systems which were not designed considering their possible growth, the possible integration of new technologies and their interconnectedness.

Critical infrastructures are required to be resilient against the risk of mismanagement errors caused by accidental faults, but their growth makes them a more and more important target for malicious attacks and intrusions. Diagnosis is hence an essential step to assure the resiliency of the infrastructure, aiming to identify the faults affecting the system components and hence trigger proper reconfiguration actions. Reasons have been given to explain why traditional diagnostic solutions are not well suited as they are: they are based on static assumptions about system behavior, fault model and detection mechanisms (the "unusual" component behavior can be defined a–priori and assumed due to faults in the component itself). The direction where searching for new methods and solutions for diagnosis has been identified, that is the capability of dealing with dynamic scenarios where failure definitions and/or system specifications change over time.

This thesis has proposed a new conceptual framework for on–line system diagnosis; the proposed framework outlines that diagnosis is based on the observation of system behavior over time (monitoring), collecting information at different architectural levels and correlating together diagnostic judgments inferred from lower architectural levels and relevant events observed somewhere in the system. The proposed framework explicitly takes into account both the local and the global point of view within the infrastructure: each system node diagnoses both itself (local diagnosis) and remote nodes (private diagnosis) based on the local perception of the behavior of the remote nodes; moreover when some relevant events happen, distributed diagnosis is run to reach consensus about the diagnosis of a remote node.

The thesis has then explored the instance of the proposed framework in three different systems, each with a different role.

The first system considered has been the SCADA–based infrastructure studied in the CRUTIAL project, where the focus has been on enhancing the protection of the ICT infrastructure underlying the power production and distribution grid. The CRUTIAL architecture has been presented, showing how the proposed framework has been integrated in it. Then some quantitative analysis has been presented, aiming to evaluate some dependability properties of the building blocks protecting the infrastructure.

The second system considered has been the highly dynamic and possibly unreliable open communication infrastructure object of the HIDENETS project; this infrastructure has been defined so that it is able to support available and resilient distributed applications and mobile services with critical requirements. The HIDENETS architecture has been presented, showing how the basics of the proposed framework has been integrated in it. Then a quantitative analysis of some failure detection and reconfigurations strategies for the management of a replicated server pool in the infrastructure has been presented.

The third system considered has been the SCADA–based infrastructure currently studied in the INSPIRE project, where the focus is on security at communication layer. The project is still ongoing, so the work within the INSPIRE infrastructure has not been completed yet. The design principles of a diagnostic mechanism for the detection of sinkhole attacks to the routing protocol of a wireless sensor network have been presented; this diagnostic mechanisms has been defined as an instance of the framework proposed in this thesis, but it still lacks the identification of the relevant information to be monitored and the tuning of the internal parameters.

The theoretical contribution of the thesis has been therefore a novel conceptual framework for diagnosis, in which a conceptual schema for automatic correlation of monitored information is integrated. This is in line with the emerging role of diagnosis in critical infrastructures, where the classical error detection activity is going to be substituted by system monitoring: this leads to both an increasing amount of information to be elaborated from the diagnosis viewpoint and an increasing complexity in the overall process of diagnosis (here comes the need for correlation). The proposed approach has been developed taking into account all the three aspects involved in the diagnosis of a component (the component itself, the deviation detection mechanism and the diagnostic mechanism) and treating them in a modular way; this modularity has favored generality and has widened the applicability of the method.

The conceptual schema has been conceived in the context of the CRUTIAL project, adopting it for designing the local diagnosis of the CIS. The overall conceptual framework has been then applied in different scenarios with different aims, so that we have given the possibility to view the diagnosis problem from several perspectives: from the diagnosis of hardware faults (e.g. faults of GPS receivers within HIDENETS) to the detection of malicious attacks (e.g. sinkhole attacks to the CTP protocol in INSPIRE). This wide perspective has given the possibility to enrich the framework and make it more general.

The quantitative evaluations performed have let us estimate the impact of the overall diagnosis process on the diagnosed system, with the impact being expressed in terms of dependability measures such as probability of system failure or unavailability. The impact has been evaluated mainly looking at the effects of the reconfiguration actions triggered by the diagnosis process, at varying parameters related to both the effectiveness of detection/diagnosis and the reconfiguration strategy in place. Evaluations have in general confirmed the positive impact of good diagnosis and have underlined the strict relationship between the diagnosis and reconfiguration activities.

We still lack a complete implementation of the framework encompassing altogether all the characteristics presented in this thesis, which could reinforce the validation of the framework itself. We plan to work in this direction in the near future: part of this work is already planned within the INSPIRE project, where the framework is going to be implemented for the correlation of monitoring information devoted to the diagnosis of malicious attacks. Here the plan is to complete the definition of the relevant indicators to be used during system monitoring, the consequent tuning the internal parameters of the diagnostic mechanism (e.g. correlation rules, probability thresholds) and the final evaluation of some dependability and/or performances measures. Other future work is envisioned in the direction of improving the support for the automatic update of the parameters internal to the diagnostic framework.

BIBLIOGRAPHY

[Abou El Kalam 07] Anas Abou El Kalam, Yves Deswarte, Amine Baïna & Mohamed Kaâniche. *Access Control for Collaborative Systems: A Web Services Based Approach.* In IEEE International Conference on Web Services, 2007. ICWS 2007., pages 1064–1071, July 2007. (Cited on page 52.)

[Ammann 88] Paul E. Ammann & John C. Knight. *Data Diversity: An Approach to Software Fault Tolerance*. IEEE Transactions on Computers, vol. C-37, no. 4, pages 418–425, April 1988. (Cited on page 90.)

[Avižienis 67] Algirdas Avižienis. *Design of fault-tolerant computers*. In AFIPS '67 (Fall): Proceedings of the November 14-16, 1967, fall joint computer conference, pages 733–743, New York, NY, USA, 1967. ACM. (Cited on page 13.)

[Avižienis 04] Algirdas Avižienis, Jean C. Laprie, Brian Randell & Carl Landwehr. *Basic concepts and taxonomy of dependable and secure computing*. IEEE Transactions on Dependable and Secure Computing, vol. 1, no. 1, pages 11–33, January 2004. (Cited on pages 7, 8, 9, and 10.)

[Barborak 93] Michael Barborak, Anton Dahbura & Miroslaw Malek. *The consensus problem in fault-tolerant computing*. ACM Comput. Surv., vol. 25, no. 2, pages 171–220, 1993. (Cited on page 36.)

[Barsi 76] Ferruccio Barsi, Fabrizio. Grandoni & Piero Maestrini. *A Theory of Diagnosability of Digital Systems*. IEEE Transactions on Computers, vol. C-25, no. 6, pages 585–593, June 1976. (Cited on page 14.)

[Bessani 07] Alysson Bessani, Paulo Sousa, Miguel Correia, Nuno Neves & Paulo Veríssimo. *Intrusion-Tolerant Protection for Critical Infrastructures*. DI/FCUL TR 07-8, Department of Informatics, University of Lisbon, April 2007. (Cited on page 55.)

[Bessani 08a] Alysson Bessani, Hans P. Reiser, Paulo Sousa, Ilir Gashi, Vladimir Stankovic, Tobias Distler, Rüdiger Kapitza, Alessandro Daidone & Rafael Obelheiro. *FOREVER: Fault/in-*

trusiOn REmoVal through Evolution & Recovery. In Companion '08: Proceedings of the ACM/IFIP/USENIX Middleware '08 Conference Companion, pages 99–101, New York, NY, USA, 2008. ACM. (Cited on pages 45 and 88.)

[Bessani 08b] Alysson Neves Bessani, Rafael R. Obelheiro, Paulo Sousa & Ilir Gashi. *On the Effects of Diversity on Intrusion Tolerance*. Technical report 08–30, Dep. of Informatics, Univ. of Lisbon, December 2008. (Cited on pages xi, 93, 94, 97, and 98.)

[Bessani 09] Alysson N. Bessani, Alessandro Daidone, Ilir Gashi, Rafael Obelheiro, Paulo Sousa & Vladimir Stankovic. *Enhancing Fault/Intrusion Tolerance through Design and Configuration Diversity*. In Proceedings of the 3rd Workshop on Recent Advances on Intrusion-Tolerant Systems (WRAITS 2009), June 2009. (Cited on page 88.)

[Bhatkar 05] Sandeep Bhatkar, Sekar R. & DuVarney Daniel C. *Efficient Techniques for Comprehensive Protection from Memory Error Exploits*. In Proc. of the 14th USENIX Security Symposium, pages 271–286, August 2005. (Cited on page 92.)

[Bolch 05] Gunter Bolch, Stefan Greiner, Hermann de Meer & Kishor S. Trivedi. Queueing networks and Markov chains. Wiley-Interscience, 2005. (Cited on page 20.)

[Bondavalli 97] Andrea Bondavalli, Silvano Chiaradonna, Felicita Di Giandomenico & Fabrizio Grandoni. *Discriminating Fault Rate and Persistency to Improve Fault Treatment*. In 27th IEEE Int. Symposium on Fault-Tolerant Computing (FTCS-27), pages 354–362, Seattle, Washington, USA, June 25-27 1997. (Cited on page 16.)

[Bondavalli 00] Andrea Bondavalli, Silvano Chiaradonna, Felicita Di Giandomenico & Fabrizio Grandoni. *Threshold-Based Mechanisms to Discriminate Transient from Intermittent Faults*. IEEE Transactions on Computers, vol. 49, no. 3, pages 230–245, 2000. (Cited on pages 14, 15, 16, and 17.)

[Bondavalli 04a] Andrea Bondavalli, Silvano Chiaradonna, Domenico Cotroneo & Luigi Romano. *Effective Fault Treatment for Improving the Dependability of COTS and Legacy-Based Applications*. IEEE Transactions on Dependable and Secure Computing, vol. 1, no. 4, pages 223–237, 2004. (Cited on pages 14 and 17.)

[Bondavalli 04b] Andrea Bondavalli, Silvano Chiaradonna, Felicita Di Giandomenico & Ivan Mura. *Dependability Modeling and Evaluation of Multiple-Phased Systems using DEEM*. IEEE Transactions on Reliability, vol. 53, no. 4, pages 509–522, 2004. (Cited on pages 65 and 95.)

[Bozinovski 07] Marjan Bozinovski, Hans-Peter Schwefel & Ramjee Prasad. *Maximum availability server selection policy for efficient and reliable session control systems*. IEEE/ACM Transactions on Networking, vol. 15, no. 2, pages 387–399, 2007. (Cited on page 124.)

[Casimiro 07] António Casimiro, Andrea Bondavalli, Mário Calha, Marius Clemetsen, Alessandro Daidone, Mônica Dixit, Zoltán Égel, Lorenzo Falai, Felicita Di Giandomenico, Audun F. Hansen, Gábor Huszerl, András Kövi, Marc-Olivier Killijian, Tom Lippmann, Yaoda Liu, Erling V. Matthiesen, Henrique Moniz, Anders Nickelsen, Jimmy J. Nielsen, Thibault Renier, Matthieu Roy, José Rufino, Hans-Peter Schwefel & Inge-Einar Svinnset. *Resilient Architecture*. HIDENETS Project Deliverable D2.1.2, December 2007. (Cited on pages 33, 105, 107, 108, and 110.)

[Castro 02] Miguel Castro & Barbara Liskov. *Practical Byzantine Fault-Tolerance and Proactive Recovery*. ACM TOCS, vol. 20, no. 4, pages 398–461, 2002. (Cited on page 94.)

[Coccoli 03] Andrea Coccoli & Andrea Bondavalli. *Analysis of Safety Related Architectures*. In WORDS 2003, 9th IEEE International Workshop on Object-oriented Real-time Dependable Systems, Capri, Italy, 2003. IEEE Computer Society Press. (Cited on page 14.)

[Daidone 06] Alessandro Daidone, Felicita Di Giandomenico, Andrea Bondavalli & Silvano Chiaradonna. *Hidden Markov Models as a Support for Diagnosis: Formalization of the Problem and Synthesis of the Solution*. In 25th IEEE Symposium on Reliable Distributed Systems (SRDS 2006), pages 245–256, Leeds, UK, October 2006. (Cited on pages 11, 14, 15, 17, 19, and 42.)

[Daidone 08] Alessandro Daidone, Silvano Chiaradonna, Andrea Bondavalli & Paulo Veríssimo. *Analysis of a Redundant Architecture for Critical Infrastructure Protection*. In Rogério De Lemos, Felicita Di Giandomenico, Cristina Gacek, Henry Muccini & Marlon Vieira, editors, Architecting Dependable Systems V, volume 5135 of *LNCS*, chaper 4, pages 78–100. Springer Berlin / Heidelberg, August 2008. (Cited on pages 45, 61, 67, 93, and 99.)

[Daidone 09] Alessandro Daidone, Andrea Bondavalli, Massimo Ficco, Simon Pietro Romano, Stefano Avallone, Luigi Coppolino, Salvatore D'Antonio & Luigi Romano. *Techniques for diagnosis and recovery of SCADA systems*. INSPIRE Project Deliverable D3.2, November 2009. (Cited on pages 143 and 146.)

[Daly 00] David Daly, Daniel D. Deavours, Jay M. Doyle, Patrick G. Webster & William H. Sanders. *Möbius: An Extensible Tool for Performance and Dependability Modeling*. In 11th International Conference, TOOLS 2000, volume Lecture Notes in Computer Science, pages 332–336, Schaumburg, IL, 2000. B. R. Haverkort, H. C. Bohnenkamp, and C. U. Smith (Eds.). (Cited on page 126.)

[Dawson 06] Robert Dawson, Colin Boyd, Ed Dawson & Juan M. González Nieto. *SKMA: a key management architecture for SCADA systems*. In ACSW Frontiers '06: Proceedings of the 2006 Australasian workshops on Grid computing and e-research, pages 183–192, Darlinghurst, Australia, 2006. Australian Computer Society, Inc. (Cited on page 46.)

[Egan 07] Matthew J. Egan. *Anticipating Future Vulnerability: Defining Characteristics of Increasingly Critical Infrastructure-like Systems*. Journal of Contingencies and Crisis Management, vol. 15, no. 1, pages 4–17, March 2007. (Cited on page 3.)

[Egel 08] Zoltan Egel, Irene de Bruin, António Casimiro, Mário Calha, Geir Egeland, Lorenzo Falai, Bjarke Freund-Hansen, Sonia Heemstra de Groot, Audun Fosselie Hansen, Gábor Huszerl, Marc-Olivier Killijian, András Kövi, Tom Lippmann, Luís Marques, Erling V. Matthiesen, Anders Nickelsen, Gergely Pintér, Matthieu Roy, Hans-Peter Schwefel, Alessandro Daidone, Gaëtan Séverac, Inge-Einar Svinnset, Christophe Zanon & Manfred Reitenspieß. *Documentation and Evaluation of the experimental work*. HIDENETS Project Deliverable D6.4, December 2008. (Cited on page 113.)

[Fischer 85] Michael J. Fischer, Nancy A. Lynch & Michael S. Paterson. *Impossibility of distributed consensus with one faulty process*.

Journal of the ACM, vol. 32, no. 2, pages 374–382, 1985. (Cited on page 51.)

[Fonseca 07] Rodrigo Fonseca, Omprakash Gnawali, Kyle Jamieson, Sukun Kim, Philip Levis & Alec Woo. *The Collection Tree Protocol (CTP)*. TEP (TinyOS Enhancement Proposal) 123, Network Working Group of TinyOS Community, February 2007. (Cited on page 143.)

[Forrest 97] Stephanie Forrest, Anil Somayaji & David H. Ackley. *Building Diverse Computer Systems*. In HOTOS '97: Proceedings of the 6th Workshop on Hot Topics in Operating Systems (HotOS-VI), pages 67–72, Cape Cod, MA, USA, 1997. IEEE Computer Society Press. (Cited on page 92.)

[Gashi 06] Ilir Gashi & Peter Popov. *Rephrasing Rules for Off-The-Shelf SQL Database Servers*. 6th European Dependable Computing Conference (EDCC-6), vol. 0, pages 139–148, October 2006. (Cited on page 90.)

[Gashi 08] Ilir Gashi & Vladimir Stankovic. List of rules for diversifying operating systems and applications and their respective runtime environment(s). available at http://www.csr.city.ac.uk/people/ilir.gashi/ConfigDiv/, 2008. (Cited on pages 90 and 92.)

[Gong 95] Li Gong, Patrick Lincoln & John Rushby. *Byzantine Agreement with Authentication: Observations and Applications in Tolerating Hybrid and Link Faults*. In Dependable Computing for Critical Applications, volume 10 of *Dependable Computing and Fault tolerant Systems*, pages 139–157. IEEE Computer Society press, 1995. (Cited on page 36.)

[Gordon 06] Lawrence A. Gordon, Martin P. Loeb, William Lucyshyn & Robert Richardson. *2006 CSI/FBI computer crime and security survey*, 2006. (Cited on page 46.)

[Gray 86] Jim Gray. *Why Do Computers Stop and What Can be Done About it?* In 5th Symposium on Reliability in Distributed Software and Database Systems (SRDSDS-5), pages 3–12. IEEE Computer Society Press, Los Angeles, CA, USA, 1986. (Cited on page 89.)

[Hansen 93] Stephen E. Hansen & E. Todd Atkins. *Automated System Monitoring and Notification With Swatch*. In LISA '93: Proceedings of the 7th USENIX conference on System administration, pages 145–152, Berkeley, CA, USA, 1993. USENIX Association. (Cited on page 24.)

[Iyer 90] Ravishankar K. Iyer, Luke T. Young & P. V. Krishna Iyer. *Automatic Recognition of Intermittent Failures: An Experimental Study of Field Data*. IEEE Transactions on Computers, vol. 39, no. 4, pages 525–537, 1990. (Cited on page 14.)

[Jecheva 06] Veselina Jecheva. *About Some Applications of Hidden Markov Model in Intrusion Detection Systems*. In International Conference on Computer Systems and Technologies, 2006. (Cited on page 20.)

[Johnson 07] Chris Johnson & Miroslaw Malek. *Progress achieved in the research area of Critical Information Infrastructure Protection by the IST-FP6 Projects CRUTIAL, IRRIIS and GRID*. Technical report, EU Report, March 2007. (Cited on page 3.)

[Lei 08] Peter Lei, Lyndon Ong & Tuexen Michael. *An Overview of Reliable Server Pooling Protocols*. Standards track, Internet Engineering Task Force (IETF), September 2008. (Cited on page 122.)

[Liang 05] Yinglung Liang, Yanyong Zhang, Anand Sivasubramaniam, Ramendra K. Sahoo, Jose Moreira & Manish Gupta. *Filtering Failure Logs for a BlueGene/L Prototype*. In DSN '05: Proceedings of the 2005 International Conference on Dependable Systems and Networks, pages 476–485, Washington, DC, USA, 2005. IEEE Computer Society. (Cited on page 39.)

[Lin 90] Ting-Ting Y. Lin & Daniel P. Siewiorek. *Error Log Analysis: Statistical Modeling and Heuristic Trend Analysis*. IEEE Transactions on Reliability, vol. 39, pages 419–432, 1990. (Cited on page 14.)

[Lollini 08] Paolo Lollini, Andrea Bondavalli, Francesco Brancati, Andrea Ceccarelli, Marius Clemetsen, Ludovic Courtès, Alessandro Daidone, Geir Egeland, Lorenzo Falai, Jesper Grønbæk, Ossama Hamouda, Audun Fosselie Hansen, Martin B. Hansen, Mohamed Kaâniche, Marc-Olivier Killijian, Máté Kovács, Melinda Magyar, István Majzik, Erling V. Matthiesen, Leonardo Montecchi, Anders Nickelsen, Jimmy J. Nielsen, David Powell, Jakob G. Rasmussen, Thibault Renier & Hans-Peter Schwefel. *Application of the evaluation framework to the complete scenario (final* *version)*. HIDENETS Project Deliverable D4.2.2, December 2008. (Cited on pages 105, 122, and 129.)

[Madani 05] Vahid Madani & Damir Novosel. *Getting a grip on the grid*. Spectrum, IEEE, vol. 42, pages 42–47, 2005. (Cited on page 46.)

[Manik 09] Miroslav Manik & Elena Gramatova. *Diagnosis of faulty units in regular graphs under the PMC model*. In DDECS '09: Proceedings of the 2009 12th International Symposium on Design and Diagnostics of Electronic Circuits&Systems, pages 202–205, Washington, DC, USA, 2009. IEEE Computer Society. (Cited on page 14.)

[Marques 09] Luís Marques, António Casimiro & Mário Calha. *Design and development of a proof-of-concept platooning application using the HIDENETS architecture*. In Proceedings of the 2009 IEEE/IFIP Conference on Dependable Systems and Networks, pages 223–228, Estoril, Lisboa, Portugal, June 2009. (Cited on pages 105 and 114.)

[Martin 06] Jean-Philippe Martin & Lorenzo Strigini. *Fast Byzantine Consensus*. IEEE Transactions on Dependable and Secure Computing, vol. 3, no. 3, pages 202–215, 2006. (Cited on page 36.)

[Meyer 85] John F. Meyer, A. Movaghar & William H. Sanders. *Stochastic Activity Networks: Structure, Behavior and Application*. In International Workshop on Timed Petri Nets, pages 106–115. IEEE Computer Society Press, July 1985. (Cited on page 126.)

[Mongardi 93] Giorgio Mongardi. *Dependable computing for railway control systems*. In DCCA-3, pages 255–277, Mondello, Italy, 1993. (Cited on pages 14 and 15.)

[Moretto 04] Marco Moretto. Progettazione, realizzazione ed utilizzo di un generatore di simulatori per sistemi a fasi multiple. Master's thesis, Università degli Studi di Pisa, 2004. (Cited on pages 69, 73, and 95.)

[Morin 97] Christine Morin & Isabelle Puaut. *A survey of recoverable distributed shared virtual memory systems*. IEEE Transactions on Parallel and Distributed Systems, vol. 8, no. 9, pages 959–969, 1997. (Cited on page 87.)

[Mura 01] Ivan Mura & Andrea Bondavalli. *Markov Regenerative Stochastic Petri Nets to Model and Evaluate the Dependability of Phased Missions*. IEEE Transactions on Computers, vol. 50, no. 12, pages 1337–1351, 2001. (Cited on pages 65, 69, and 95.)

[Nickelsen 09] Anders Nickelsen, Jesper Grønbæk, Thibault Renier & Hans-Peter Schwefel. *Probabilistic Network Fault-Diagnosis Using Cross-Layer Observations*. In International Conference on Advanced Information Networking and Applications, volume 0, pages 225–232, Los Alamitos, CA, USA, 2009. IEEE Computer Society. (Cited on pages 14 and 17.)

[Nitzberg 91] Bill Nitzberg & Virginia Lo. *Distributed shared memory: a survey of issues and algorithms*. Computer, vol. 24, no. 8, pages 52–60, August 1991. (Cited on page 87.)

[Obelheiro 06] Rafael Obelheiro, Alysson. Bessani, Lau C. Lung & Miguel Correia. *How Practical Are Intrusion-Tolerant Distributed Systems?* DI/FCUL TR 06–15, Department of Informatics, University of Lisbon, 2006. (Cited on pages 45 and 62.)

[Pizza 98] Michele Pizza, Lorenzo Strigini, Andrea Bondavalli & Felicita Di Giandomenico. *Optimal Discrimination between Transient and Permanent Faults*. In 3rd IEEE High Assurance System Engineering Symposium, pages 214–223, Bethesda, MD, USA, 1998. (Cited on pages 15 and 17.)

[Porcarelli 01] Stefano Porcarelli, Felicita Di Giandomenico, Amine Chohra & Andrea Bondavalli. *Tuning of Database Audits to Improve Scheduled Maintenance in Communication Systems*. In U. Voges, editor, SAFECOMP 2001, 20th Int. Conference on Computer Safety, Reliability and Security, pages 238–248, Budapest, Hungary, 2001. Springer-Verlag. (Cited on page 112.)

[Porcarelli 04] Stefano Porcarelli, Marco Castaldi, Felicita Di Giandomenico, Andrea Bondavalli & Paola Inverardi. *A Framework for Reconfiguration-based Fault-Tolerance in Distributed Systems*. In R. De Lemos, c. Gacek & A. Romanovsky, editors, Architecting Dependable Systems, LNCS. Springer-Verlag, 2004. (Cited on pages 111 and 112.)

[Powell 98] David Powell, Christophe Rabéjac & Andrea Bondavalli. *Alpha-count mechanism and inter-channel diagnosis*. Technical report, ESPRIT Project 20716 GUARDS Report, N°I1SA1.TN.5009.E, 1998. (Cited on pages 17 and 36.)

[Powell 99] David Powell, Jean Arlat, Ljerka Beus-Dukic, Andrea Bondavalli, Paolo Coppola, Alessandro Fantechi, Eric Jenn, Christophe Rabéjac & Andy Wellings. *GUARDS: A Generic Upgradable Architecture for Real-Time Dependable Systems*. IEEE Transactions on Parallel and Distributed Systems, vol. 10, no. 6, pages 580–599, 1999. (Cited on page 37.)

[Powell 03] David Powell & Robert Stroud. *Conceptual Model and Architecture of MAFTIA*. MAFTIA Project Deliverable D21, January 2003. (Cited on page 33.)

[Preparata 67] Franco P. Preparata, Gernot Metze & Robert T. Chien. *On the Connection Assignment Problem of Diagnosable Systems*. IEEE Transactions on Electronic Computers, vol. EC-16, no. 6, pages 848–854, December 1967. (Cited on pages 13 and 14.)

[Prewett 04] James E. Prewett. *Listening to your cluster with LoGS*. In The 5th LCI International Conference on Linux Clusters: the HPC revolution 2004, May 2004. (Cited on page 24.)

[Pucella 06] Riccardo Pucella & Fred B. Schneider. *Independence From Obfuscation: A Semantic Framework for Diversity*. In Proc. of the 19th IEEE Workshop on Computer Security Foundations, pages 230–241, 2006. (Cited on page 92.)

[Rabiner 90] Lawrence R. Rabiner. *A tutorial on hidden Markov models and selected applications in speech recognition*. In Alex Waibel & Kai-Fu Lee, editors, Readings in speech recognition, pages 267–296. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990. (Cited on pages 20 and 42.)

[Romano 02] Luigi Romano, Andrea Bondavalli, Silvano Chiaradonna & Domenico Cotroneo. *Implementation of Threshold-based Diagnostic Mechanisms for COTS-based Applications*. In 21st IEEE Symposium on Reliable Distributed Systems (SRDS'02), pages 296–303, Osaka University, Suita, Japan, October 13-16 2002. (Cited on pages 12 and 17.)

[Rosenberg 02] Jonathan Rosenberg, Henning Schulzrinne, Gonzalo Camarillo, Alan Johnston, Jon Peterson, Robert Sparks, Mark Handley & Eve Schooler. *SIP: Session Initiation Protocol.*

Standards track, Internet Engineering Task Force (IETF), June 2002. (Cited on page 122.)

[Rouillard 04] John P. Rouillard. *Real-time Log File Analysis Using the Simple Event Correlator (SEC)*. In LISA '04: Proceedings of the 18th USENIX conference on System administration, pages 133–150, Berkeley, CA, USA, 2004. USENIX Association. (Cited on pages 24 and 27.)

[Sanders 91] William H. Sanders & J.F. Meyer. *A Unified Approach for Specifying Measures of Performance, Dependability and Performability*. In A. Avizienis & J. Laprie, editors, Dependable Computing for Critical Applications, Vol. 4 of Dependable Computing and Fault-Tolerant Systems, pages 215–237. Springer Verlag, 1991. (Cited on page 72.)

[Serafini 07] Marco Serafini, Andrea Bondavalli & Neeraj Suri. *Online Diagnosis and Recovery: On the Choice and Impact of Tuning Parameters*. IEEE Transactions on Dependable and Secure Computing, vol. 4, no. 4, pages 295–312, 2007. (Cited on pages 14 and 17.)

[Siewiorek 98] Daniel P. Siewiorek & Robert S. Swarz. Reliable computer systems (3rd ed.): design and evaluation. A. K. Peters, Ltd., Natick, MA, USA, 1998. (Cited on pages 33 and 116.)

[Simoncini 04] Luca Simoncini, Felicita Di Giandomenico, Andrea Bondavalli & Silvano Chiaradonna. *Architectural Challenges for a Dependable Information Society*. In Building the Information Society, pages 282–304. Springer Boston, 2004. (Cited on page 3.)

[Sosnowski 94] Janusz Sosnowski. *Transient Fault Tolerance in Digital Systems*. IEEE Micro, vol. 14, no. 1, pages 24–35, 1994. (Cited on page 15.)

[Sousa 06] Paulo Sousa, Nuno Neves, Antonia Lopes & P. Veríssimo. *On the Resilience of Intrusion-Tolerant Distributed Systems*. DI/FCUL TR 6–14, Department of Informatics, University of Lisbon, 2006. (Cited on page 52.)

[Sousa 07] Paulo Sousa, Alysson Bessani, Miguel Correia, Nuno Neves & Paulo Veríssimo. *Resilient Intrusion Tolerance through Proactive and Reactive Recovery*. In 13th IEEE Pacific Rim Dependable Computing conference, 2007. (Cited on pages 52, 53, 61, 63, and 73.)

[Sousa 08] Paulo Sousa, Alysson N. Bessani & Rafael R. Obelheiro. *The FOREVER Service for Fault/Intrusion Removal*. In 2nd Workshop on Recent Advances on Intrusion-Tolerant Systems (WRAITS'08). ACM, April 2008. (Cited on page 88.)

[Spainhower 92] Lisa Spainhower, Jack Isenberg, Ram Chillarege & Joseph Berding. *Design for Fault-Tolerance in System ES/9000 Model 900*. In Twenty-Second International Symposium on Fault-Tolerant Computing (FTCS-22) Digest of Papers, pages 38–47, Jul 1992. (Cited on pages 14 and 15.)

[Task Force 04] Task Force. August 14th blackout: Causes and recommendations. U.S.-Canada Power System Outage Task Force, April 2004. (Cited on page 2.)

[Tendolkar 82] Nandakurnar N. Tendolkar & Robert L. Swann. *Automated Diagnostic Methodology for the IBM 3081 Processor Complex*. IBM J. Research and Development, vol. 26, pages 78–88, 1982. (Cited on page 15.)

[Thompson 04] Kerry Thompson. *Logsurfer*. SysAdmin magazine, vol. 2004-3, pages –, March 2004. Publication has ceased in August 2007. (Cited on page 24.)

[UCTE 06] UCTE. *Lessons learnt from the disturbance on 4 November 2006*. In Marcel Bial, editor, UCTE Annual Report, chaper 5, pages 22–27. Secretariat of UCTE, 2006. (Cited on page 1.)

[Vaarandi 02] Risto Vaarandi. *SEC - A Lightweight Event Correlation Tool*. In IEEE Workshop on IP Operations and Management, pages 111–115, 2002. (Cited on page 24.)

[Vedeshenkov 02] V. A. Vedeshenkov. *On the BGM Model-Based Diagnosis of Failed Modules and Communication Lines in Digital Systems*. Automation and Remote Control, vol. 63, no. 2, pages 316–327, February 2002. (Cited on page 14.)

[Veríssimo 06a] Paulo Veríssimo. *Travelling through wormholes: a new look at distributed systems models*. ACM SIGACT News, vol. 37, no. 1, pages 66–81, 2006. (Cited on pages 33, 54, 88, and 106.)

[Veríssimo 06b] Paulo Veríssimo, Nuno Neves, Christian Cachin, Jonathan Poritz, David Powell, Yves Deswarte, Robert Stroud & Ian Welch. *Intrusion-Tolerant Middleware: The Road to Automatic*

Security. IEEE Security and Privacy, vol. 4, no. 4, pages 54–62, Jul./Aug. 2006. (Cited on page 50.)

[Veríssimo 08] Paulo Veríssimo, Nuno Neves, Miguel Correia, Anas Abou El Kalam, Yves Deswarte, Andrea Bondavalli & Alessandro Daidone. *The CRUTIAL Architecture for Critical Information Infrastructures*. In Rogério De Lemos, Felicita Di Giandomenico, Cristina Gacek, Henry Muccini & Marlon Vieira, editors, Architecting Dependable Systems V, volume 5135 of *LNCS*, chaper 1, pages 1–27. Springer Berlin / Heidelberg, August 2008. (Cited on pages 33, 45, 47, and 54.)

[Walter 97] Chris J. Walter, Patrick Lincoln & Neeraj Suri. *Formally Verified On-Line Diagnosis*. IEEE Transactions on Software Engineering, vol. 23, no. 11, pages 684–721, 1997. (Cited on pages 31, 34, and 35.)

[Wilson 06] Clay Wilson. *Terrorist capabilities for cyber-attack*. In Myriam Dunn & Victor Mauer, editors, Int. CIIP Handbook volume II, volume 2 of *International CIIP handbook*, chaper 2, pages 69–88. CSS, ETH Zurich, 2006. (Cited on page 46.)

[Zhou 02] Lidong Zhou, Fred Schneider & Robbert Van Rennesse. *COCA: A Secure Distributed Online Certification Authority*. ACM TOCS, vol. 20, no. 4, pages 329–368, November 2002. (Cited on page 94.)

ACRONYMS

AH	Authentication Header.
API	Application Programming Interface.
BN	Bayesian Network.
CI	Critical Infrastructure.
CIS	CRUTIAL Information Switch; a CIS is a special interconnection and filtering device which connects the WAN to the LANs in the CRUTIAL architecture.
COST	European Co-Operation in Science and Technology; it is an intergovernmental framework for European cooperation in the field of scientific and technical research. `http://www.cost.esf.org/`
COTS	Commercial Off–The–Shelf.
CRUTIAL	CRitical UTility InfrastructurAL Resilience; it is an IST project (IST-2004-27513) which addressed new networked ICT systems for the management of the electric power grid. `http://crutial.erse-web.it/`
CS	Computer Society.
CTP	Control Tree Protocol.
DEEM	DEpendability Evaluation of Multiple–phased systems; it is a dependability modeling and evaluation tool specifically tailored for the time–dependent analysis of MPS. `http://dcl.isti.cnr.it/DEEM/`
DoS	Denial of Service.
DSPN	Deterministic and Stochastic Petri Net.
DTMC	Discrete Time Markov Chain.
FCR	Fault Containment Region.
FOREVER	Fault/intrusiOn REmoVal through Evolution & Recovery; FOREVER is a recent project funded by the EU through the RESIST NoE (Contract IST-2004-26764). `http://forever.di.fc.ul.pt/`

FP6	Sixth Framework Programme; it is a framework programme for research and technological development funded by the European Commission between 2002 and 2006.
FP7	Seventh Framework Programme; it is a framework programme for research and technological development funded by the European Commission between 2007 and 2013.
FRU	Fault Replacement Unit.
GPS	Global Positioning System; it is a U.S. space–based global navigation satellite system and provides reliable positioning, navigation, and timing services to worldwide users on a continuous basis.
GRID	GRID is a coordination action funded by the IST-FP6. `http://grid.jrc.it/`
GSPN	Generalized Stochastic Petri Nets.
GUARDS	Generic Upgradable Architecture for Real-time Dependable Systems; it is an ESPRIT Project (contract n. 20716) which addressed the development of architectures, methods, techniques, and tools to support the design, implementation and validation of critical real–time systems. `http://www.cs.york.ac.uk/rts/projects/guards/guards.html`
HIDENETS	HIghly DEpendable ip–basedNETworks and Services; it is an IST project (IST-2004-26979) which developed and analyzed end–to–end resilience solutions for distributed applications and mobility-aware services in ubiquitous communication scenarios. `http://www.hidenets.aau.dk/`
HMM	Hidden Markov Model; it is an extension of Markov models.
ICR	Intrusion Containment Region.
ICT	Information and Communications Technology.
IEEE	Institute of Electrical and Electronic Engineers; IEEE is an international non–profit, professional organization for the advancement of technology related to electricity.
IFIP	International Federation for Information Processing; IFIP is an umbrella organization for national societies working in the field of information technology.

INSPIRE	INcreasing Security and Protection through Infrastructure REsilience; it is an IST STReP (ICT-SEC-FP7-225553) aiming at increasing the security and the protection of SCADA–based infrastructures at communication layer. `http://www.inspire-strep.eu/`
IntelliCIS	Intelligent Monitoring, Control and Security of Critical Infrastructure Systems; it is a COST action (IC0806) aiming at developing innovative intelligent monitoring, control and safety methodologies for critical infrastructure systems. `http://www.intellicis.eu/`
IP	Internet Protocol.
IPsec	Internet Protocol Security ; it is a protocol suite for securing Internet Protocol (IP) communications.
IRRIIS	Integrated Risk Reduction of Information–based Infrastructure Systems; it is an IST–FP6 project which worked for increasing the availability, survivability and resilience of critical power and communication infrastructures. `http://www.irriis.org/`
IST	Information Society Technologies; it is one of the thematic priorities in the research funded by the European Commission.
LAN	Local Area Network.
LIFO	Last In, First Out.
LQI	Link Quality Indication.
MAC	Message Authentication Code.
MAFTIA	Malicious–and Accidental–Fault Tolerance for Internet Applications; it is a FP5 research project (IST-1999-11583) which investigated the dependability of large distributed applications. `http://research.cs.ncl.ac.uk/cabernet/www.laas.research.ec.org/maftia/`
MPS	Multiple Phased System.
MTTF	Mean Time To Failure.
NMEA	National Marine Electronics Association.
NVD	National Vulnerability Database.

OSI	Open System Interconnection reference model; it is an abstract description for layered communications and computer network protocol design.
PRRW	Proactive–Reactive Recovery Wormhole; it is a service defined in the CRUTIAL architecture.
QoS	Quality of Service.
RMS	Root Mean Square.
RSSI	Received Signal Strength Indication.
RTU	Remote Terminal Unit.
SAN	Stochastic Activity Network.
SCADA	Supervisory Control And Data Acquisition; it is a category of industrial control system.
SEC	Security.
SRN	Stochastic Reward Nets.
SSL	Secure Sockets Layer protocol.
SVM	Shared Virtual Memory.
TC	Technical Committee.
TCP	Transmission Control Protocol.
THL	Time Has Lived.
TMR	Triple Modular Redundancy.
TLS	Transport Layer Security protocol.
TSO	Transmission System Operator.
UCTE	Union for the Coordination of Transmission of Electricity; `http://www.ucte.org/`
WAN	Wide Area Network.
WSN	Wireless Sensor Network.