Project no.:	**IST-FP6-STREP - 027513**
Project full title:	**Critical Utility InfrastructurAL Resilience**
Project Acronym:	**CRUTIAL**
Start date of the project:	**01/01/2006 Duration: 36 months**

Deliverable no.: D25

Title of the deliverable: Model-based evaluation of the middleware services and protocols & architectural patterns

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

Contractual Date of Delivery to the CEC:	**31/12/2007**
Actual Date of Delivery to the CEC:	**18/01/2008**
Organisation name of lead contractor for this deliverable: CNIT	
Author(s): Susanna Donatelli[6](Editor), Eric Alata[4], Andrea Bondavalli[3], Davide Cerotti[6], Alessandro Daidone[3], Silvano Chiaradonna[3], Felicita DiGiandomenico[3], Mohamed Kaâniche[4], Vincent Nicomette[4], Francesco Romani[3], Luca Simoncini[3]	
Participant(s):[6]CNIT, [3]CNR-ISTI, [4]LAAS-CNRS.	
Work package contributing to the deliverable:	WP5
Nature:	R
Dissemination level:	PU
Version:	3.0
Total number of pages:	

Abstract

This task has the objective of providing an evaluation of the service and protocols defined in WP4, as well as defining reusable models, for dependability evaluation approach (incl. Probabilistic verification). This task will also take care of defining the metrics that are adequate to evaluate interdependencies and to compute the values of such metrics on a number of reference systems. This task goes hands in hands with task T3.3 in WP3. We shall model system architecture blocks, threats and interdependencies with the goal of assessing the ability of the deployed architectural solutions to limit the negative effects of interdependencies. The output of this task is a quantitative analysis to support designers in the activity of defining a robust Information Infrastructure (II) for Electric Power Systems

Keyword list: performance and dependability evaluation, architectural building blocks, system verification, Electrical Power System evaluation

DOCUMENT HISTORY

Date	Version	Status	Comments
29/10/2007	000	Draft	First version of table of contents
14/12/2007	001	draft	Includes contribution from ISTI, UNIFI, CNIT, LAAS
18/12/2007	001	draft	Correctness of material included has been checked by partners and editor has added some material to link sections and some questions to partners
4/1/2008	001	draft	Contribution back to editor
16/1/2008	002	draft	Almost final integrated version distributed
18/1/2008	003	submit	Submitted version

Table of Contents

1 INTRODUCTION AND OUTLINE

This deliverable reports on a number of evaluation and validation activities that have been carried on in the project through the use of models using the formalisms, methodologies and tools introduced in deliverables D8 [Kaâniche *et al.* 2008] and D11[Donatelli *et al.* 2008b]. They are quite diverse activities, but highly representative of the various roles that model-based approaches can play in the analysis of critical infrastructure. At first (Section 2), we set up the field with the definition of the metrics considered in the analysis reported in this deliverable, we then (Section 3) introduce a model of attacks which is a probability distribution of the time between two successive attacks, fitted upon the data collected through honeypots. This model can be used as an input to all building blocks and models that include attack situations. Section 4 describes the analysis of a portion of the CRUTIAL middleware: the CIS intrusion tolerant block described and built in within WP4 (See Deliverable D10 [Neves *et al.* 2008]. We have attacked the analysis from two points of view: correctness of the proposed solution (Section 4.1), and evaluation of the system failure probability under various configurations of CIS, in particular for the case in which rejuvenation techniques are applied within the CIS (Section 4.2). The third target of the analysis is the evaluation of the interdependencies between the electrical infrastructure EI and the information infrastructure II, which is reported in Section 5. The deliverable concludes with Section 6, that discusses the progress of model based analysis in CRUTIAL, while identifying the planned work for the next period.

2 PRELIMINARIES

Before introducing the model-based analysis of the three targets above, we set the objectives of the analysis and we update some of the UML description of CRUTIAL upon which the scenarios are based.

2.1 Metrics definition

This subsection provides an overview of the properties evaluated and on the metrics computed: it provides therefore an overview of the contribution of the research reported in this deliverable.

2.1.1 Metrics for quantitative evaluation

To quantify the effects of interdependencies, appropriate metrics need to be defined. Measures of performability, a unified measure proposed to deal simultaneously with performance and dependability, are particularly helpful in risk analysis of the Electrical Power System (EPS) based on a stochastic approach. The EPS corresponds to the composition of the electrical infrastructure (EI) and the Information and control infrastructures (denoted as II). A set of measures specific for the EPS can be based on the following reward structure where costs and rewards are considered with respect to the point of the view of the power producers and distributors:

- To each generator a cost is associated, depending on the generated power, the type of generator, the breakdown of the generator;
- To each load a positive reward is associated, depending on the consumed power and on the criticality of the load;
- To each interruption of service supply a cost is associated, depending on the difference between the required power and the available power for each load, on the number of loads which will be powered off, on the criticality of loads which will be powered off, and on the duration of the interruption.

Among the performability measures that have been defined to assess the impact of interdependencies between the Electric and the Information infrastructures of EPS, the following ones are currently adopted in the EPSyS simulator (see Deliverable D11 [Donatelli *et al.* 2008b]).

- The expected reward $E[V_t]$ at time t, defined as:

$$V_t = \sum_{P \in RS} R(P) I_t$$

where I_t is a random variable that is equal to 1 if the real injected power in the transmission grid at time t is P, otherwise $I_t = 0$, and R(P) is the reward associated to the real injected power P, where:

$$R(P) = \sum_{i \in G} P_i W_i^G + \sum_{j \in L} P_j W_i^L$$

with W^G_i and W^L_j the cost and the reward associated to generator i or load j, respectively.

- The expected reward $E[Y_{[0,t]}]$ accumulated in the interval [0, t], with Y[0,t] defined as:

$$Y_{[0,t]} = \sum_{P \in RS} R(P) J_{[0,t]}$$

where J[0,t] is a random variable that represents the total time that real injected power has value P during the time interval [0, t].

- The expected percentages of blackout B_t and $B_{[0,t]}$ at time t and in the interval [0, t], respectively. This is a particular case of the previous ones.
- The expected numbers of components N_t and $N_{[0,t]}$ affected by a disruption at time t and in the interval [0, t], respectively.

Traditionally, performability measures have been used to assess the impact of accidental threats. A similar approach could be adopted to assess the impact of malicious threats provided that representative assumptions about the occurrence of attacks and their impact can be defined. The results presented in Section 3 aim to fulfil this objective. In particular, we focus on the characterization of the random variable of the times between successive attacks observed at a target system connected to the Internet (e.g., a honeypot). The definitions of an "Attack" , and the "Time between attacks" are given as follows:

- An attack is defined by the set of packets exchanged between a source identified by an Internet IP address and a particular target system (e.g. a honeypot).
- Let us denote by *T* the random variable corresponding to the time between the occurrence of two consecutive attacks at a given target, and *t* a realization of *T*. The random variable T can be characterized by different metrics such as, its probability distribution function *F(t)*, its probability density function denoted as *f(t)* or its expected value, denoted as *E(T)*:

$$F(t) = Prob.\{T \le t\}$$

$$f(t) = \frac{dF(t)}{dt}$$

$$E(T) = \int_0^\infty t f(t) dt$$

With reference to the evaluation and validation of CRUTIAL architectural solutions, relevant measures of interest are those representative of the system resilience, since the devised mechanisms/protocols are intended to reinforce the ability of the considered infrastructures to survive despite accidental faults as well as malicious attacks. Therefore, indicators such as the system failure probability and the system availability are among the relevant measures for this purpose. Actually, these two measures have been selected and evaluated in the study focusing on the Proactive-Reactive Recovery strategy proposed to reinforce the intrusion tolerance of the CIS in the scope of the protection service (see Section 4.2)

2.1.2 Metrics for correctness analysis

The objective of the analysis is to "prove", through model-checking, that CIS is correct according to the specification given in [Sousa *et al.* 2007], where correctness is defined through the following properties:

1. *If f+1 correct replicas call approve function for the same message then it will be signed*.
2. *If a legal message is received by some correct replica and the message was not previously forwarded by other replicas, it will eventually be signed*.

From validity of these lemmas follows correctness and integrity:

3. *A legal message received by at least one replica is forwarded to its destination.*
4. *An illegal message is never processed by its destination*.

2.2 Extending the UML representation of the CRUTIAL domain

The UML diagrams proposed in [Garrone et al. 2007] mainly concern the general architecture of the both the Electrical Infrastructure (EI) and the Information Infrastructure (II) (also referred to as ICT in this section). Such diagrams were obtained by structuring the information extracted from WP1 activities, in the form of UML class diagrams. Actually interdependencies between infrastructures do not emerge explicitly from those UML diagrams. The further step in the UML representation of the CRUTIAL project domain, is justified by the necessity of making interdependencies explicit; to this aim, UML collaboration diagrams and state diagrams have been exploited. In this activity, we focused on the services that the II provides to the EPS; services are represented as collaboration constructs and collaboration occurences (instances).

In UML, a **collaboration** describes a structure of collaborating elements (roles), each performing a specialized function which collectively accomplishes the desired functionality. Given the high level of abstraction no identity or precise classes of participants need to be defined [UML]. A **collaboration occurrence** (instance) represents the application of the pattern described by a collaboration to a specific situation involving precise classes or objects playing the roles of the collaboration.

As an example, we consider the teleoperation service where three roles can be identified: a *remote controller* sending commands to a *local controller* in charge of managing a *controlled entity*. Such roles are represented by the collaboration called *Teleoperation* in the collaboration diagram in Figure 2-1.a. Another example of collaboration is depicted in the collaboration diagram in Figure 2-1.b where we consider the *Communication* between two nodes of a communication channel whose roles are *Peer A* and *Peer B* respectively.

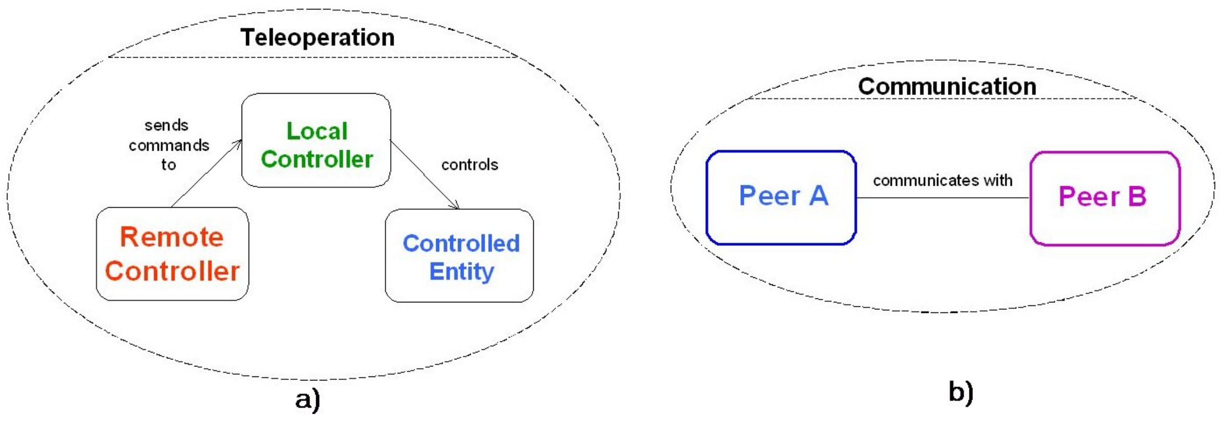


Figure 2-1: a) collaboration *Telecontrol* b) collaboration *Communication*

The teleoperation service may be exploited in order to change the configuration of the GRID; in this case, a teleoperation command is sent from an *Area Control Centre* to a *Substation Automation Site* controlling a *Substation*. This can be represented by instancing the collaboration called *Teleoperation* (Figure 2-1.a) to a more specific situation where the role of remote controller is played by the *Area Control Centre*, the role of *local controller* is played by the *Substation Automation Site*, and the role of *controlled entity* is played by the *Substation*. Such instantiation is represented by the collaboration diagram in Figure 2-2 where the collaboration *Teleoperation* is connected by means of dashed arcs to the objects playing the corresponding roles. Still in Figure 2-2, the collaboration called *Communication* (Figure 2-1.b) is istanced twice in order to express the presence of two communication channels: between *Area Control Centre* (*Peer A*) and *Substation Automation Site* (*Peer B*); between *Substation Automation Site* (*Peer A*) and *Substation* (*Peer B*).

Moreover, in the diagram in Figure 2-2, ICT elements can be distinguished from EPS elements by means of the stereotypes <<ICT>> and <<EPS>> labelling the objects. The presence of an association arc between an ICT element and an EPS element, puts in evidence the presence of an interdependency between the ICT and the EPS infrastructure.

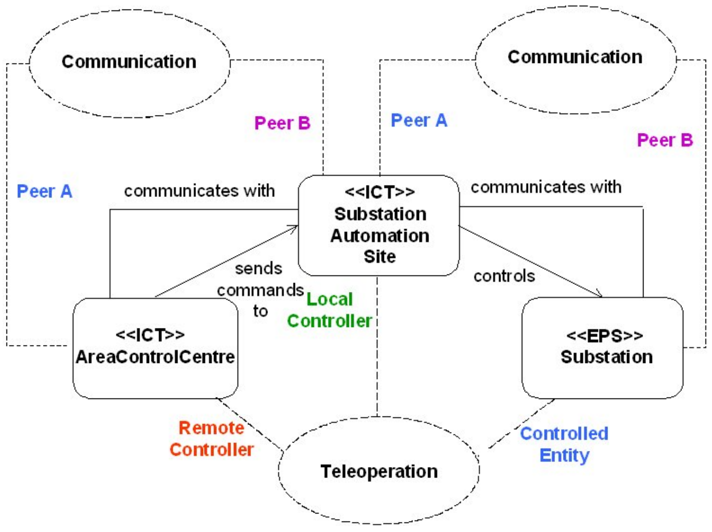


Figure 2-2: instances of the collaboration *Telecontrol* (Figure 2-1.a) and of the collaboration *Communication* (Figure 2-1.b)

The area control centre and the substation automation site are elements present in the control system scenario n. 1 described in [Garrone et al. 2007]; in that case, the sites are still connected by communication channels. Therefore the collaborations in Figure 2-1.a and in Figure 2-1.b can be instanced to the objects describing the scope of such control system scenario; this is shown in Figure 2-3 corresponding to Figure 5-9 of [Garrone et al. 2007], with the addition of the collaboration instances.

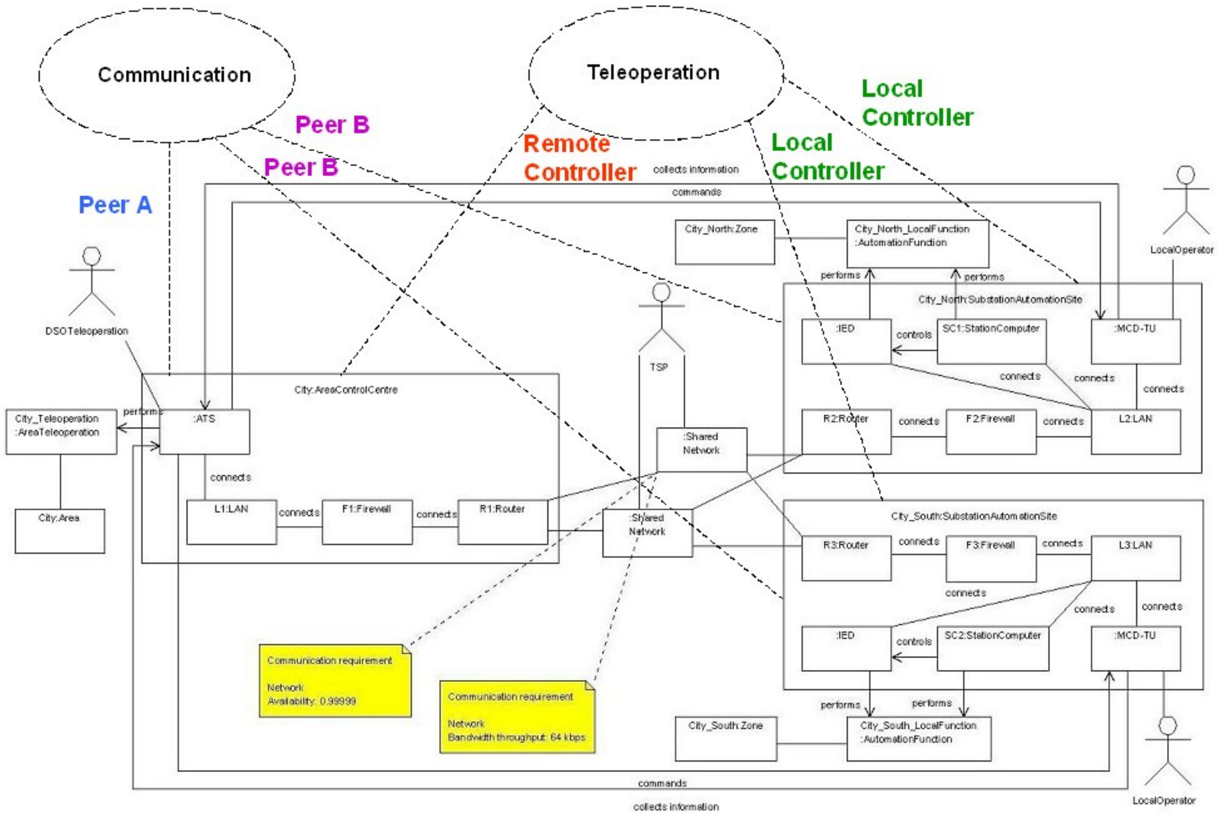


Figure 2-3: the object diagram of the control system scenario n.1 [Garrone et al. 2007] with the addition of the instances of the collaboration *Teleoperation* (Figure 2-1.a) and of the collaboration *Communication* (Figure 2-1.b)

Each service implemented by a collaboration is also described by a state diagram, and state transitions depend on the actual states of the collaboration components (for structured collaborations) or on the state of the objects that play the specific roles.

For example, the states of *Communication* are represented by the diagram in Figure 2-4.a, where the possible states are *Idle*, *Normal*, *Delayed* and *FailedDelivery*. In such diagram, the state transitions are due to the success of a DoS attack or to the success of the countermeasures [Garrone et al. 2007]. The states of *Teleoperation* are instead represented in the state diagram in Figure 2-4.b, where the possible states are *Normal*, *PartialLoss* and *CompleteLoss*. The state transitions in Figure 2-4.b are due to the current state of *Communication* according to the state diagram shown in Figure 2-4.a. In this way, we represent the interdependency between *Teleoperation* (EPS) and *Communication* (ICT).

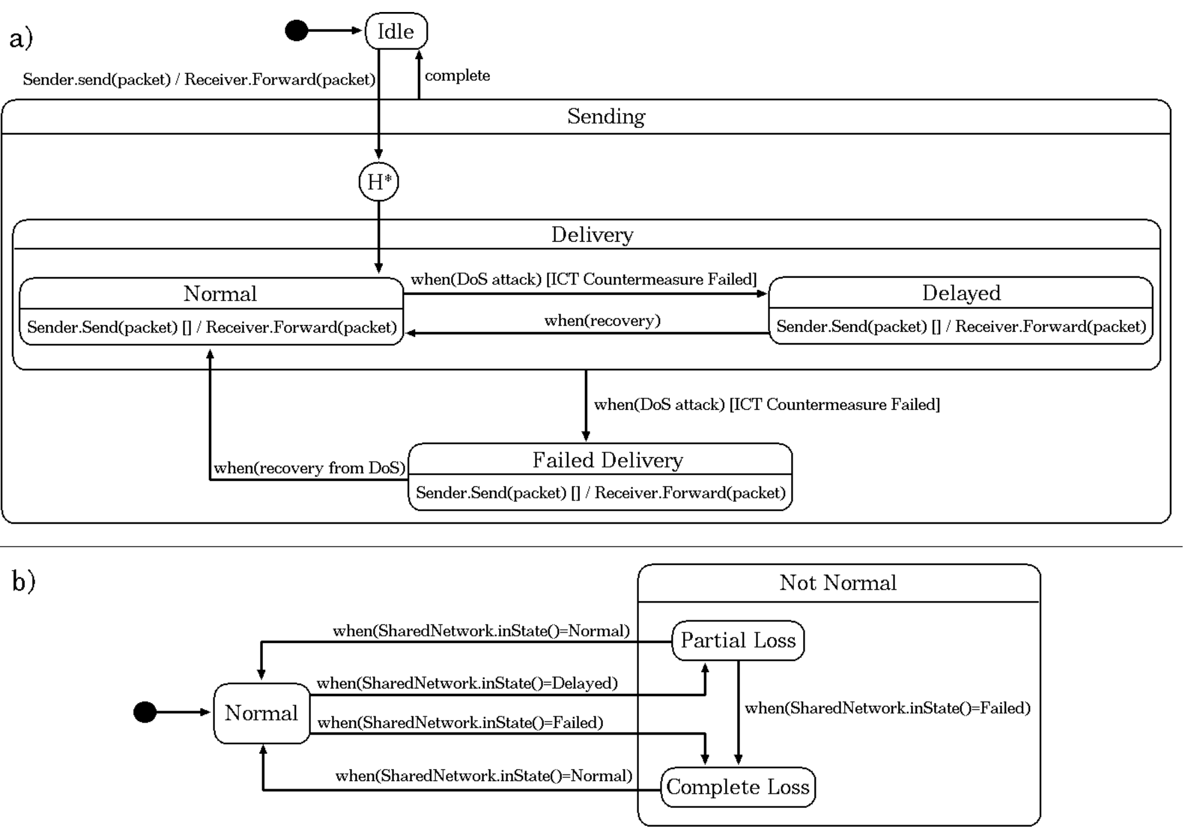

a)
Idle
Sender.send(packet) / Receiver.Forward(packet)
complete
Sending
H*
Delivery
Normal
Sender.Send(packet) [] / Receiver.Forward(packet)
when(DoS attack) [ICT Countermeasure Failed]
when(recovery)
Delayed
Sender.Send(packet) [] / Receiver.Forward(packet)
when(DoS attack) [ICT Countermeasure Failed]
when(recovery from DoS)
Failed Delivery
Sender.Send(packet) [] / Receiver.Forward(packet)
b)
Not Normal
when(SharedNetwork.inState()=Normal)
Partial Loss
when(SharedNetwork.inState()=Delayed)
Normal
when(SharedNetwork.inState()=Failed)
when(SharedNetwork.inState()=Failed)
when(SharedNetwork.inState()=Normal)
Complete Loss

Figure 2-4: a) state diagram of *Communication* (Figure 2-1.a) b) state diagram of *Teleoperation* (Figure 2-1.b)

3 STATISTICAL MODELS OF ATTACKS

This section deals with the elaboration of statistical models that are representative of malicious traffic observed on the Internet using data collected from honeypots. Such models are very relevant to establish realistic assumptions about the distribution and the intensity of internet attacks targeting systems and infrastructures connected through the Internet, as in CRUTIAL. Also, they constitute a first step towards the elaboration of stochastic models aimed at evaluating quantitative measures characterizing the impact of malicious threats on the target systems, as discussed in the context of CRUTIAL modelling methodology addressed in Workpackage 2.

As detailed in deliverable D26, honeypots have been increasingly used in the recent years to collect real data about malicious traffic on the Internet. A honeypot is a machine connected to the Internet that no one is supposed to use and whose value lies in being probed, attacked or compromised [Spitzner 2002]. The statistical models of attacks discussed in this section are based on the data collected from the *Leurré.com* environment, which is a cooperative attack data collection initiative set up by Eurecom to which LAAS contributes, based on distributed honeypot platforms [Pouget *et al.* 2005]. This environment integrates up to eighty identically configured low-interaction honeypots that have been deployed progressively since 2003.

Deploying honeypots at a distributed and large scale is interesting to collect a large volume of data characterizing malicious activities observed at various locations of the Internet. One of the questions that can be raised is whether data collected by honeypots deployed at different locations exhibit similar or different phenomena and whether the attack processes observed show different or similar statistical distributions.

The results presented in this section are aimed at addressing these questions. The objective is to elaborate analytical statistical models that faithfully reflect the distribution of the interarrival time between attacks observed at various honeypot platforms. Such models provide useful insights about the statistical characteristics of malicious traffic observed on the Internet. They can be used to generate synthetic workloads that are representative of malicious traffic. The statistical distributions presented in this section are also useful to support the definition of quantitative evaluation models based on realistic assumptions.

This section is organized as follows. Section 3.1 gives an overview of the collected data and of some of the problems that need to be addressed to exploit the data for building models. Section 3.2 presents the proposed methodology. Section 3.3 deals with the statistical modelling of the times between attacks based on the data collected from the deployed honeypots and presents some examples of results. Finally, Section 3.4 discusses future work.

3.1 Overview of the Leurré.com environment and the collected data

The *Leurré.com* data collection environment is aimed at deploying at various geographical locations on the Internet a large set of identically configured low interaction honeypot platforms using the freely available software called *honeyd* [Provos *et al.* 2007]. The objective is to collect a large volume of data that can be used to carry out representative and non biased analyses of attack processes. Each platform emulates three computers running Linux RedHat, Windows 98 and Windows NT, respectively, and various services such as ftp, web, etc. A firewall ensures that connections cannot be initiated from the computers, only replies to external solicitations are allowed. All the honeypot platforms are centrally managed to ensure that they have exactly the same configuration. The data gathered by each platform are securely uploaded to a centralized database with the complete content, including payload of all packets sent to or from these honeypots, and additional information to facilitate its analysis, such as the IP geographical localization of packets' source addresses, the OS of the attacking machine, the local time of the source, etc.

The data recorded in the database can be analyzed at various levels of granularities. Indeed, the packets received at each platform can be grouped e.g. according to the source address, the target virtual machine, the time between the arrival of consecutive packets received from the same source, etc.

The concepts of « source » and « attack » used in this section are defined as follows :

- A *source* corresponds to an IP address observed on one or many platforms, for which the inter-arrival time between two consecutive packets does not exceed a given threshold (25 hours). The time difference is computed by converting all times to GMT (Greenwich Mean Time).
- An *attack* is composed by the set of packets exchanged between a source and a particular honeypot platform.

The deployment of the honeypots has been carried out progressively starting in 2003. To date, up to 80 honeypot platforms have been deployed at various locations in academia and industry, in 30 countries, covering the five continents. The total number of attacks recorded in the Leurré.com database between February 2003 and August 2007 is 4 873 564 attacks issued from 3 026 972 different IP addresses. This constitutes a significantly large sample on which statistical analyses can be performed.

Table 1 gives some statistics summarizing the number of attacks observed on each platform. It can be seen that the level of malicious activity recorded on the different platforms was not uniform. This can be explained to some extent by the fact that the platforms have been deployed progressively as illustrated in Figure 3-1.

Table 1 - Statistics on the number of attacks recorded on the honeypot platforms

Min	Max	Average	Median	Std. deviation
3	504651	62480.59	39594.5	81140.93

Figure 3-1 gives an overview of the deployment where each bar associated with a given platform indicates the time interval between the first packet and the last packet recorded on the platform. As can be seen from the figure, the observation period of the different platforms was not uniform. Some of them have been operational only for a short period of time, compared to others for which we have data covering 4 years. It is important when performing comparative analysis of attack processes observed on several platforms that all the platforms have been observed during the same sufficiently long period of time.

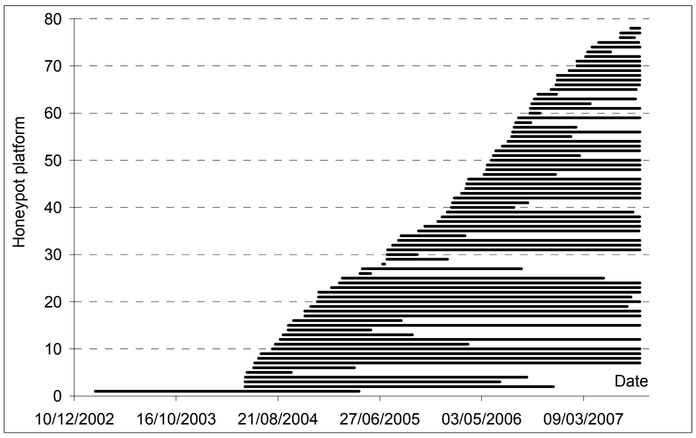


Figure 3-1 - Number of deployed honeypots evolution

Considering Figure 1 again, the observation period indicated by the bar associated with each platform does not mean that the platform was active all the time during this period. Indeed, for several reasons, some of the honeypots exhibited many times, *silence periods* during which no activity was recorded. These silence periods are more likely due to the unavailability or the unreachability of the honeypot from the Internet as a consequence of power failures, network failures or simply due to the disconnection of the honeypot itself for administration and maintenance activities. Two examples are presented in Figure 3-2 and Figure 3-3 which plot the evolution of the number of attacks per day recorded on Platforms 9 and 37 respectively.

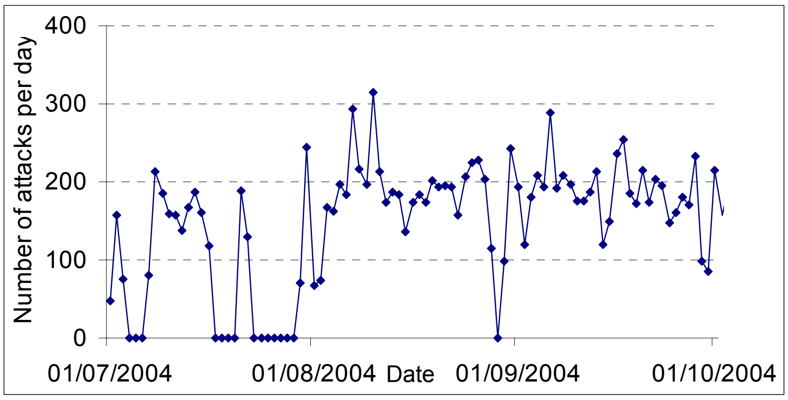


Figure 3-2- Evolution of the number of attacks per day observed on platform 9

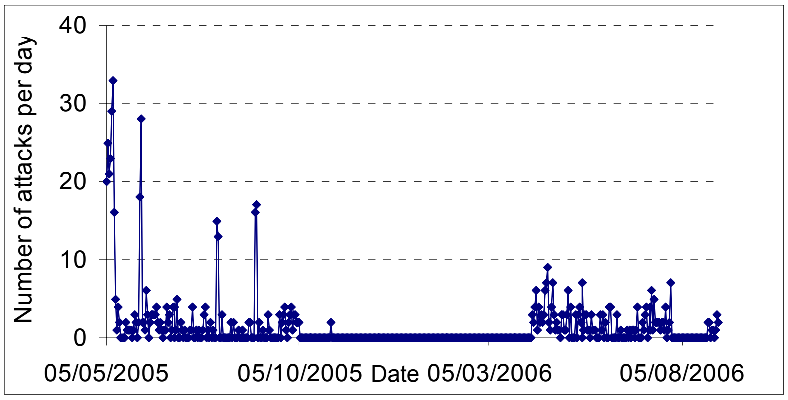


Figure 3-3 - Evolution of the number of attacks per day observed on platform 37

Considering the intensity of attacks observed on average per day on each platform, it is more likely that the silence periods are more related to unreachability problems than to the absence of activities from the attackers. Thus, it is important to identify and process such periods before building models characterizing the occurrence of attacks, otherwise the results will be biased. In the following section, we present the methodology that we have developed to address this problem.

3.2 Methodology

The methodology that we have set up to deal with silence periods consists of two main steps:

1) identification of the silence periods,
2) selection of the data observation period and the platforms to be included in the modeling of the distribution of times between attacks based on the results obtained in step 1.

3.2.1 Identification of silence periods

In our data, the silence periods generally correspond to atypical and infrequent intervals of time between attacks that are significantly separated in value from the rest of the other observations recorded on the honeypot platform. Accordingly, they can be considered as "outliers".

Various statistical tests exist for the identification of outliers, e.g., Nixon, Grubbs or boxplot tests [Hodge *et al.* 2004]. In our methodology, we used the modified boxplot test defined in [Vanderviere *et al.* 2004], which is well suited when the distribution of the data is skewed, which is the case of our honeypot data. This test proceeds in two steps.

At a first step, this test computes for the considered data set D a metric denoted as MC(D), taking values in the interval [-1, 1], that measures the skewness of the distribution. Positive (respectively, negative) values correspond to positively (respectively, negatively) skewed distributions, and when MC(D) is null the distribution is not skewed.

At a second step, the test computes a critical interval that depends on the sign of the skewness metric MC(D), such that any value outside this interval is considered as an outlier.

Let us denote by Q1, Q2, and Q3 the first, second and third quartiles of the considered data sample D, and let IQR = Q3-Q1. The test identifies outliers as follows:

If $MC(D) \geq 0$, x is an outlier $\Leftrightarrow$
$x \notin [Q1-1.5\, e^{-4MC(D)}\, IQR;\ Q3 +1.5\, e^{3MC(D)} IQR]$

If $MC(D) < 0$, x is an outlier $\Leftrightarrow$
$x \notin [Q1-1.5\, e^{-3MC(D)}\, IQR;\ Q3 +1.5\, e^{4MC(D)} IQR]$

We have applied this test to the data collected from each honeypot platform. The percentage of identified outliers for each platform is generally less than 1%. However, we have observed a large variation of the magnitude of the intervals of time considered as outliers. The average value is around 6 hours and the standard deviation is about 78 hours, considering the values of the outliers identified for the 80 platforms.

3.2.2 Data selection for the modelling of inter-arrival times between attacks

The outliers identified in the first step of our methodology correspond to suspicious periods of silence. In our context, we make the assumption that they most likely correspond to unavailability periods of the corresponding platform, than to periods of deliberate inactivity of the attackers.

Then, the question is: what should we do with these outliers? Usually, two solutions are investigated:

1) Remove the outliers from the data set or substitute them by synthetic values generated based on the general characteristics of the sample distribution.
2) Select a subset of the initial data such that the impact of the outliers is reduced.

The first solution is not acceptable in our context as it might lead to biased results. Also it makes the comparison of the attack processes observed on different platforms considering the same period of time, more difficult in particular, when the outliers correspond to long periods of time. Thus, the second solution is more suitable to our context.

In our methodology, we have considered three main criteria to select the period of time and the subset of data to be used for the statistical modelling of the times between attacks on the different honeypot platforms.

1) The length of the observation period.
2) The number of platforms included in the analysis.
3) The minimum level of average availability estimated for each platform.

The average availability of each platform is estimated based on the assumption that the silence periods correspond to unavailability periods as explained in the beginning of this section.

We have developed an iterative algorithm based on a sliding window that starts first by considering the whole data collection period and estimates the availability of each platform. If the number of platforms satisfying the minimum availability per platform criterion is higher than a predefined threshold, the algorithm stops. Otherwise, we consider a shorter period of 1 hour less and run the algorithm again until it converges.

Figure 3-4 presents graphically the results obtained from the algorithm. Some numerical examples extracted from the figure are reported in Table 2. As expected, if one sets a predefined number of platforms to be selected, increasing the minimum availability requirement to be satisfied by each platform, will lead to a shorter observation period, and vice-versa.

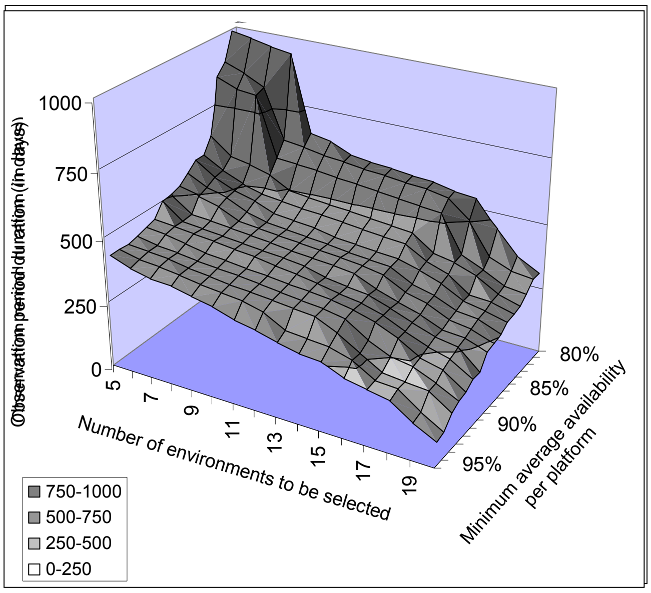


Figure 3-4- Relationship between the duration of the selected period, the minimum average availability per platform and the number of platforms satisfying the availability requirement

Table 2- Examples of results extracted from **Figure 3-4**

		Number of selected platforms		
		8	**15**	**20**
Minimum availability per platform	80%	637	448	420
	85%	490	413	343
	90%	455	350	259
	95%	287	189	89

In our study, we have set as an objective to have the longest possible observation period with a reasonable number of platforms to enable comparative analyses of attack processes observed on various platforms. Accordingly, we have selected 8 platforms with a minimum availability requirement of 80% corresponding to an observation period of 637 days. The number of platforms selected is sufficient to make significant comparative analyses.

Table 3 reports some statistics characterizing the activities observed on the selected platforms. The first column identifies the platform, the second column gives the number of intervals between attacks (#ti) observed for this platform. The following columns indicate the values of Q1, Q2 and Q3 quartiles, the maximum, the mean, and the standard deviation of the interarrival times between attacks. Finally, the last column gives the average availability of the corresponding platform. These platforms are geographically located in six different European countries: France, Italy, Belgium, Poland, Germany and UK.

Table 3- Statistics on the activities observed on the selected platforms during the observation period of 637 days

Honeypot	#ti	Q1(ti) (sec)	Q2(ti) (min)	Q3(ti) (min)	Max (ti) (min)	Mean (ti) (min)	Std. Deviation (min)	Average Availability (%)
9	134161	56	3	8	53	6	7	93
13	15742	538	32	73	382	52	59	89
14	42670	107	7	24	158	18	25	85
28	10200	578	35	96	650	73	100	82
31	90580	76	4	11	52	8	9	81
32	65962	161	7	16	76	11	12	84
42	38826	102	7	25	278	19	29	84
62	25042	435	19	40	199	29	30	80

3.3 Time between attacks distribution

Considering the selected platforms and the data collected during the selected period of time identified by the algorithm presented in the previous section, we have investigated candidate statistical models that are representative of the distribution of times between attacks observed on the different platforms.

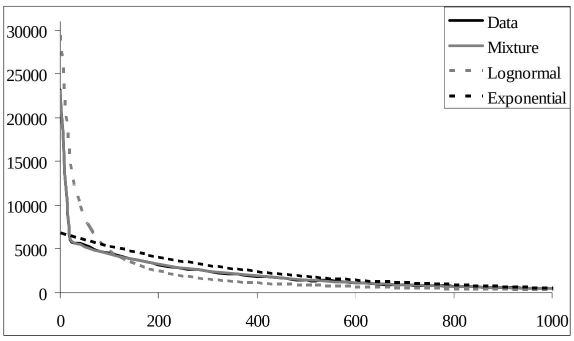
Finding tractable analytical models that faithfully reflect the observed times between attacks is useful to characterize the observed attack processes and to find appropriate indicators that can be used for prediction purposes. We have investigated several candidate distributions, including Weibull, Lognormal, Pareto, and the Exponential distribution, which are traditionally used in reliability studies. The best fit for each platform has been obtained using a mixture model combining a generalized Pareto and a Weibull distribution.

Let us denote by *T* the random variable corresponding to the time between the occurrence of two consecutive attacks at a given platform, and *t* a realization of *T*. Assuming that the probability density function *f(t)* associated to *T* is characterized by a mixture distribution combining a generalized Pareto distribution and a Weibull distribution, then f(t) is defined as follows.

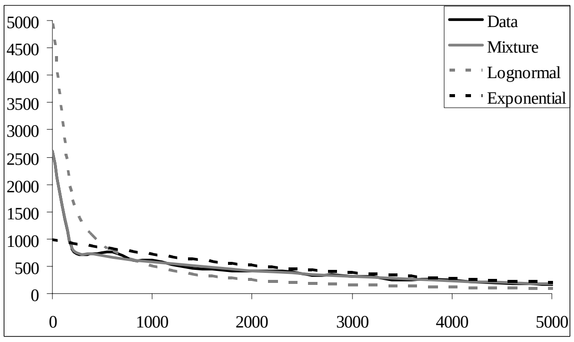
$$pdf(t) = \Pi \cdot \frac{1}{\sigma} \cdot (1 - \frac{\varepsilon t}{\sigma})^{\frac{1}{\varepsilon}-1} + (1-\Pi) \cdot \frac{k}{\lambda} \cdot (\frac{t}{\lambda})^{k-1} \cdot e^{-(\frac{t}{\lambda})^k}$$

σ and ε the parameters of the generalized Pareto distribution, *k* and λ are the parameters associated to the Weibull distribution and Π is a mixture probability.

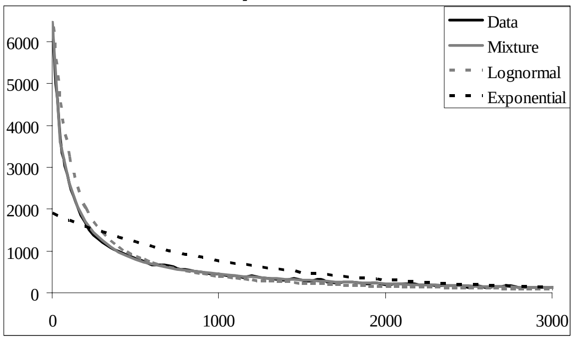
We have used the R statistical package [11] to estimate the parameters that provide the best fit to the collected data. The quality of fit is assessed by applying the Kolmogorov-Smirnov and the Chi-Squared statistical tests. The results obtained for four of the eight platforms are presented in Figure 3-5. Similar conclusions have been observed for the other platforms as well. It can be noticed that for all the platforms, the mixed distribution provides a good fit to the observed data whereas the exponential and lognormal distributions are not suitable to describe the observed attack processes. Thus, the traditional assumption considered in reliability evaluation studies assuming that failures occur according to a Poisson process does not seem to be satisfactory when considering the data observed from our honeypots.



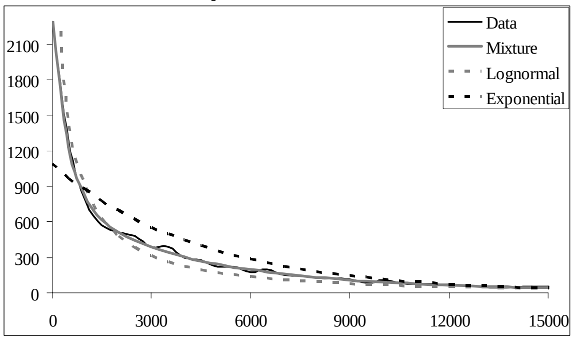
a) Platform 9



b) Platform 13



c) Platform 14



d) Platform 28

Figure 3-5- Observed and estimated times between attacks probability density functions.

As regards the interpretation of the mixture distribution, the Pareto part models the bursty arrival of attacks (correlated and intensive attacks targeting one IP address) whereas the Weibull part describes background uncorrelated attacks that occur less frequently in time.

3.4 Conclusion

In this section, we presented experimental results based on data collected during a four year observation period from a large set of identically configured honeypots that have been deployed on the Internet. In particular, we proposed a methodology to process the collected data in order to identify statistical distributions that best characterize the times between attacks observed on the different platforms, taking into account the possible presence of silence periods that are due e.g., to the unavailability of the honeypots. The experimental results show that a mixture Pareto and Weibull distribution is well suited to describe the attack processes observed on several honeypot platforms. This result confirms the preliminary investigations derived in [Kaâniche *et al.* 2006] based on a small subset of the data presented in this section. The obtained results should be useful to generate synthetic workloads that are representative of malicious traffic observed on the honeypots and to support the elaboration of quantitative security assessment models.

4 PRELIMINARY EVALUATION OF ARCHITECTURAL SOLUTIONS

In this section we analyze the CIS solution proposed and implemented in WP4. Initially we consider a CIS implementation without rejuvenation, and we prove it correct, next we analyze the performability of CIS when using a rejuvenation schema.

4.1 PRISM models of CIS

In this section we present a first attempt to model the reference architecture proposed to protect critical infrastructures. In this architecture a critical information infrastructure is formed by facilities interconnected by a wide area network; each facility is viewed as a set of connected LANs, interconnected to each other through a WAN, forming the so called WAN-of-LANs model. The protection of LANs from the WAN or other LANs is realized by the *Crutial Information Switch*(CIS) device, it acts like a firewall: it captures packets that pass through it, checks if they satisfy the security policy and either forwards the packets or discards those that do not satisfy the policy; moreover CIS provides access control capability, intrusion tolerant techniques and other services. A more detailed description of the CIS and its characteristics can be found in deliverable D4 [Abou El Kalam *et al.* 2007].

The aim of the model is to verify two basic properties of validity and integrity; the satisfaction of these properties entails that only messages in accordance with the security policy will be forwarded to their destination. We describe how this can be achieved using the tool PRISM and its model description language.

4.1.1 Description of the system

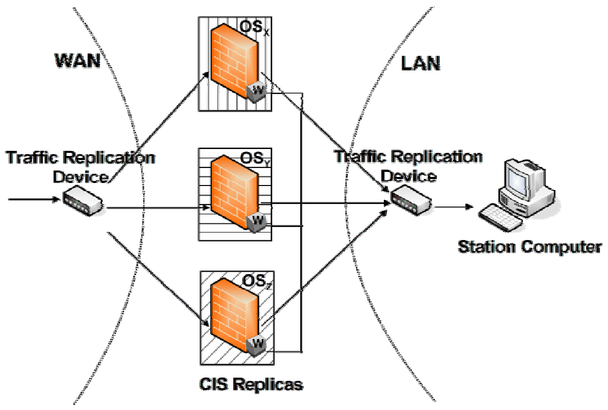


Figure 4-1: Intrusion tolerant CIS architecture.

The CIS system implements a distributed replicated firewall between a non trusted WAN and the trusted LAN that we want to protect. A replication device located at the end of the WAN multicasts to the incoming messages *n* CIS replicas. Each CIS replica verifies whether the message is in accordance with the security policy and does a vote. All CIS replicas exchange with each others their vote through point-to-point reliable channels. Only messages positively voted by at least *f+1* replicas are signed with a shared key and then forwarded to the LAN; to avoid traffic multiplication a replica is selected randomly to forward the approved message. The station computer located in the LAN will only accept messages with a valid signature.

The CIS replica is composed of two parts the payload and the wormhole; by assumption each part can be affected by different types of failure:

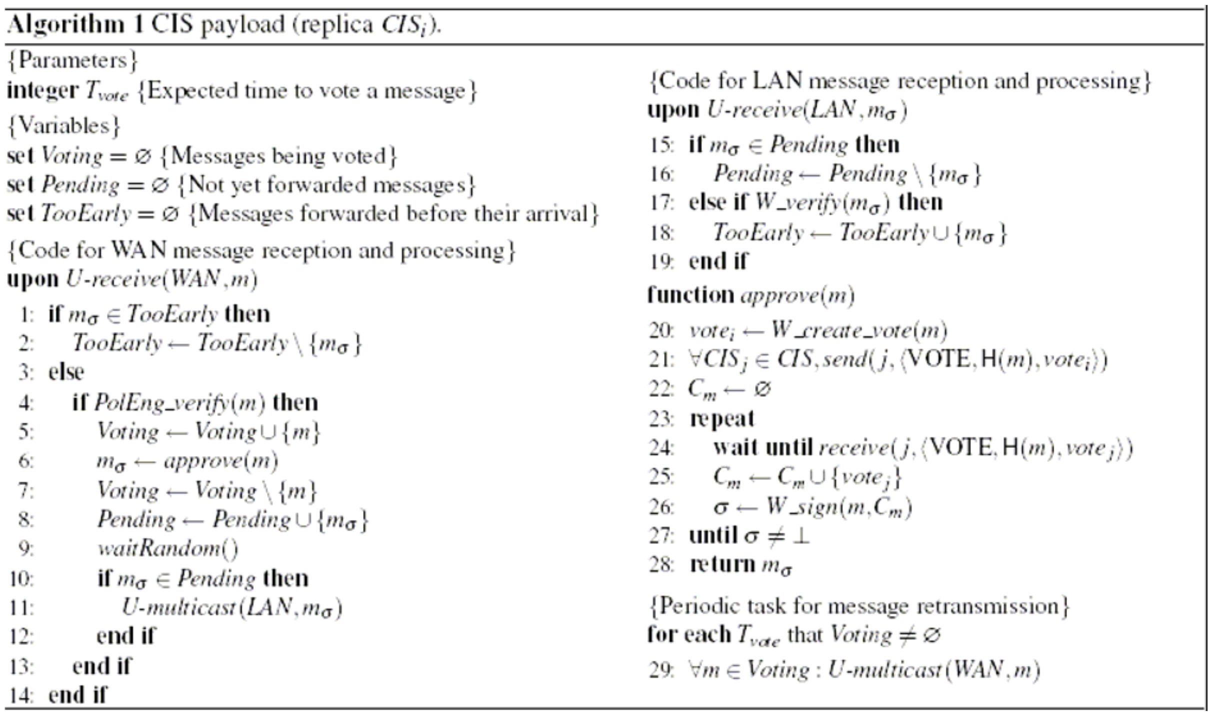
- Payload: Asynchronous system with $n \geq 2f+1$ replicas in which at most f can be subject to Byzantine failures. Fault independence is assumed for the replicas, i.e., the probability of a replica to be compromised is independent of another replica failure.

- Wormhole: secure tamperproof subsystem $W = \{W_1, ..., W_n\}$ in which at most $f_c \leq f$ local wormholes can fail by crash. It is assumed that when a local wormhole W_i crashes, the corresponding payload replica CIS_i crashes together.

Messages arriving at CIS replicas both from the WAN and the LAN have unreliable fair multicast semantics: if a message is multicasted infinitely many times it will be received by all its receivers infinitely many times.

The algorithm for processing incoming messages is shown in Figure 4-2. T_{vote} is the single configuration parameter of the payload protocol and it defines the expected time required to receive, vote and sign a legal message. Additionally, the algorithm uses three variables: *Voting*, the set of messages being voted; *Pending*, the set of messages received and approved by the replicas that were already signed by the wormhole but not yet forwarded to the station computer and *TooEarly*, the set of correctly signed messages forwarded by some other replica but not yet received (from the WAN) by the replica. There are some primitives that need to be specified:

- *U-multicast(G,m)* and *U-receive(G,m)*: the former is invoked to multicast a message *m* to the group *G*, the latter is used to receive message *m* previously multicasted to *G*, where *G* can be either WAN or LAN;
- *W_create_vote(m)* authenticates vote message *m* with a key shared between the wormholes;
- *W_Sign(m,C_m)* signs message *m* if and only if the replica payload presents a certificate set C_m containing at least *f+1* valid votes produced by different wormholes;
- *W_verify(m)* tests if the signed message *m* is valid.



Algorithm 1 CIS payload (replica CIS_i).

```
{Parameters}
integer $T_{vote}$ {Expected time to vote a message}
{Variables}
set $Voting = \emptyset$ {Messages being voted}
set $Pending = \emptyset$ {Not yet forwarded messages}
set $TooEarly = \emptyset$ {Messages forwarded before their arrival}
{Code for WAN message reception and processing}
upon $U\text{-}receive(WAN, m)$
if $m_\sigma \in TooEarly$ then
    $TooEarly \leftarrow TooEarly \setminus \{m_\sigma\}$
else
    if $PolEng\_verify(m)$ then
        $Voting \leftarrow Voting \cup \{m\}$
        $m_\sigma \leftarrow approve(m)$
        $Voting \leftarrow Voting \setminus \{m\}$
        $Pending \leftarrow Pending \cup \{m_\sigma\}$
        $waitRandom()$
        if $m_\sigma \in Pending$ then
            $U\text{-}multicast(LAN, m_\sigma)$
        end if
    end if
end if

{Code for LAN message reception and processing}
upon $U\text{-}receive(LAN, m_\sigma)$
if $m_\sigma \in Pending$ then
    $Pending \leftarrow Pending \setminus \{m_\sigma\}$
else if $W\_verify(m_\sigma)$ then
    $TooEarly \leftarrow TooEarly \cup \{m_\sigma\}$
end if
function $approve(m)$
$vote_i \leftarrow W\_create\_vote(m)$
$\forall CIS_j \in CIS, send(j, \langle \text{VOTE}, \text{H}(m), vote_i \rangle)$
$C_m \leftarrow \emptyset$
repeat
    wait until $receive(j, \langle \text{VOTE}, \text{H}(m), vote_j \rangle)$
    $C_m \leftarrow C_m \cup \{vote_j\}$
    $\sigma \leftarrow W\_sign(m, C_m)$
until $\sigma \neq \perp$
return $m_\sigma$
{Periodic task for message retransmission}
for each $T_{vote}$ that $Voting \neq \emptyset$
$\forall m \in Voting : U\text{-}multicast(WAN, m)$
```

Figure 4-2: CIS algorithm.

4.1.2 Model description

From the description in the previous section, it should be clear that the correctness of the algorithm, i.e. that only messages in accordance with the security policy will be forwarded to their destination, is the result of the interaction between different non-trivial sequences of events. This motivates the need for the construction and analysis of a formal model. The model represents only synchronization aspects (message approving) and concurrency, it does not represent cryptographic aspects.

The algorithm is composed by three code blocks: *U-receive(WAN,m)*, *U-receive(LAN,m)* and message retransmission. Name blocks preceded by keyword **upon** are executed by a new thread created when a particular event happens. For example, whenever a message *m* originating from the WAN is received by a CIS replica, a new thread will be generated that processes message *m*. The algorithm needs synchronization mechanisms that manage the concurrent access to variables shared between threads, for example the *Voting* set. Such mechanisms are not necessary for algorithm correctness therefore they are not specified. We highlight that variables are shared only between threads generated by the same CIS replica, not between threads of different CIS replicas.

In the PRISM language, a model description comprises a number of modules, each corresponding to a component of the system being modelled. Conceptually the model is composed of four templates of module: SendWANNormalMsg, SendWANMaliciousMsg, WanToCIS and Replica. The first and second templates represent a legal or malicious sender that invokes multicast of a message to the WAN. Templates WanToCIS models an unreliable multicast of message to CIS replica: a message can be lost by some or all CIS replicas. Finally, template Replica describes the states, and transitions between them, of Payload: normal, affected by Byzantine failure or crashed; and the states and transitions of Wormhole: normal or crashed. Moreover it describes the processing of a message when it is received from the WAN (function *U-receive(LAN,m)*) and from the LAN (*U-receive(WAN,m)*). We choose to model together these functions because they share the same variables.

From these templates we instantiate, through module renaming provided by PRISM, a different module for each pair *(r,t)* of replica and thread; the module describes the behaviour of thread *t* in replica *r*. We assume that every thread manages only a specific message, therefore threads are identified by the number of messages that they manage. In more detail: to describe in replica 1 the behaviour of the thread that receives from the WAN message number 2, we write a module called *Replica_1_2*, where the first number indicates replica and the second number indicates the message or equivalently the thread that processes it.

In the following we present a detailed description of Replica module template. The initial part of a module definition lists a set of finite-ranging variables which determine the possible states that the module can be in. The first variable of the Replica module is *PayS*, which keeps track of the current state of Payload: when *PayS*=0, it is in normal state; when *PayS*=1, it is affected by a Byzantine failure; when *PayS*=2, Payload is down. Variable *WormS* keeps track of state of Wormhole: when *WormS*=0, it is in normal state; when *WormS*=1, it is affected by a crash failure. Variable *ProcS* represents the state of processing of the message, variable *Voting*, *Pending* and *TooEarly* have the same meaning of the algorithm.

Set variables such as *Voting* or *Pending* can be represented as an array, unfortunately PRISM language doesn't support this structured type, therefore to model a set, say *Voting*, of *n* elements we need to define *n* variables *Voting_1*,…,*Voting_n* that indicate whether message *i* has been voted. Note that variables are shared between threads, but not between replicas, therefore we need to add another index to indicate to which replica the variable *Voting_i* belongs. The concurrent access to variables shared between threads is modelled in a simple way. For example, the behaviour specified in lines 15-16 of the algorithm can be modelled with a single PRISM command:

(WaitLANMsg21=1) & (Pending21=1) -> 1.0: (Pending21'=0) & (WaitingInLANMsg21'=0);

The first condition of the guard tests if there is in the LAN buffer of replica 2 the message number 1, the second condition tests if message 1 belongs to Pending set of replica 2 (line 15). When both conditions are satisfied, we delete message 1 from Pending set of replica 2 (line 16) and from the LAN buffer. In PRISM the test on the guard and the update of variables is an atomic action, therefore we don't need any trick to manage concurrent accesses.

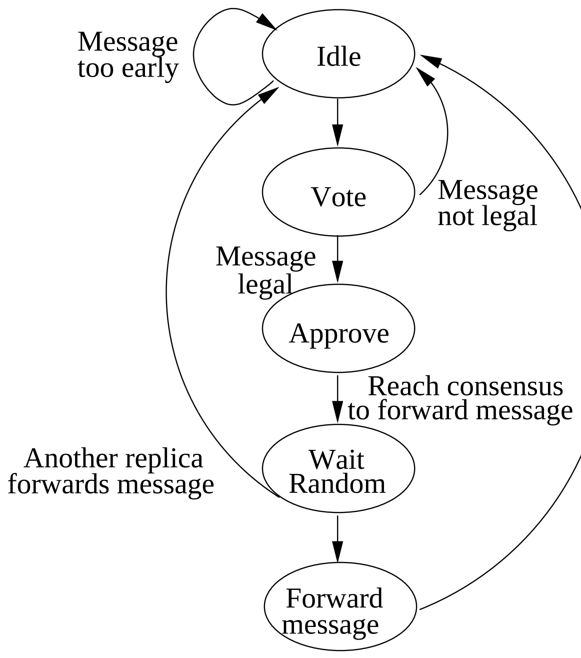


Figure 4-3: state diagram of message processing.

State diagram in Figure 4-3 summarises the steps necessary to process a message received from the WAN. The state of processing of a message is represented by variable *ProcS,* indexed with the number of the message and the number of the replica. Initially the process is idle, whenever a new message arrives we test if it belongs to *TooEarly* set, in this case the thread ignores the message and stays in idle state, otherwise it starts the voting phase. In this phase it is possible to have a normal or byzantine behaviour of the replica depending on the correspondent state of the Payload. In case of compromised Payload we assume that function *PolEng_verify(m)* deterministically returns a malicious result. In the model, whenever variable PayS assume value 0 a legal message will be positively voted and an illegal one will be negatively voted, otherwise if variable PayS assumes value 1 we will obtain the opposite result. This result is maintained in variable *Vote*. Only when for the same message at least two replicas has value 1 for variable Vote, this message will be forwarded. In the description of the system the point-to-point channel used for exchanging votes is assumed to be reliable, therefore we decide to not model it, instead the results of voting is an information accessible to all modules through global variables *Vote*. This behaviour is represented by the following command:

```
[] (majority)&(Voting11>0) & (ProcS11=2)->1.0: (ProcS11'=3)&(Voting11'=Voting11-1)&
(Sign11'=1) & (Pending11'=1);
```

where formula majority is true when at least two replicas has value 1 for variable Vote. The execution of this command deletes message from *Voting* and adds it to *Pending* set, keeps track that message is signed through updating of *Sign*, and goes to the next state WaitRandom. In this state the process waits a random amount of time, then passes to ForwardMessage state; if another replica forwarded the message we return to Idle state, otherwise we forward the message to its destination.

The language construct **if-then-else** in lines 15-19 is represented with the following instructions:

(WaitLANMsg21=1)&(Pending21=1)->1.0 : (Pending21'=0) & (WaitLANMsg21'=0);
(WaitLANMsg21=1)&(Pending21=0)&(MsgT1=1)->1.0:(TooEarly21'=1)&WaitLANMsg21'=0);
(WaitLANMsg21=1)&(Pending21=0) & (MsgT1=0) -> 1.0 : (WaitLANMsg21'=0);

We previously described the first command. The second command tests if there is in the LAN the message and if it doesn't belong to *Voting* set (otherwise we do updates in first command) and if it is positively verified (condition *MsgT1=1*); when all conditions are satisfied we add message to *TooEarly* set (update*TooEarly21'=1*) and delete it from buffer. Otherwise we simply delete it with the third command. Collectively the three commands model the behaviour of line 15-19; all constructs **if-then-else** are modelled in a similar way, however a little care is needed for nesting **if-then-else**.

```
Ctmc
        // Status of Msg 1
        global MsgT1 : [0..1] init 0;
        // 0 Msg 1 not legal
        // 1 Msg 1 legal

        global WaitingInWANMsg11 : [0..1] init 0;
        // 0 Msg 1 not sent
        // 1 Msg 1 sent to WAN

        global WaitingInLANMsg11 : [0..1] init 0;
        // 0 Msg 1 not sent
        // 1 Msg 1 sent to LAN

        global Buf11 : [0..1] init 0;
        // 0 Msg 1 is not in WAN buffer of replica 1
        // 1 Msg 1 is in WAN buffer of replica 1

        global Buf21 : [0..1] init 0;
        // 0 Msg 1 is not in WAN buffer of replica 2
        // 1 Msg 1 is in WAN buffer of replica 2

        global Buf31 : [0..1] init 0;
        // 0 Msg 3 not sent
        // 1 Msg 3 sent to WAN

        const int MaxVoteSet = 5;
        const int MaxVoted = 2;

formula majority =(  ( ((Vote11>0)?1:0) + ((Vote21>0)?1:0) + ((Vote31>0)?1:0) )>1);
formula upProcess= (PayS1=0|PayS1=1);

module SendWANNormalMsg1
[] (WaitingInWANMsg11=0)&(WaitingInWANMsg21=0)&(WaitingInWANMsg31=0) -> 1.0:
(WaitingInWANMsg11'=1)&(WaitingInWANMsg21'=1)&(WaitingInWANMsg31'=1)&(MsgT1'=1);
endmodule

module SendWANMaliciousMsg1
[] (WaitingInWANMsg11=0)&(WaitingInWANMsg21=0)&(WaitingInWANMsg31=0) -> 1.0:
(WaitingInWANMsg11'=1)&(WaitingInWANMsg21'=1)&(WaitingInWANMsg31'=1)&(MsgT1'=0);
endmodule

module WanToCIS
[] (WaitingInWANMsg11=1) -> 1.0 : (Buf11'=1)&(WaitingInWANMsg11'=0)+1.0:(WaitingInWANMsg11'=0);
[] (WaitingInWANMsg21=1) -> 1.0 : (Buf21'=1)&(WaitingInWANMsg21'=0)+1.0:(WaitingInWANMsg21'=0);
[] (WaitingInWANMsg31=1) -> 1.0 : (Buf31'=1)&(WaitingInWANMsg31'=0)+1.0:(WaitingInWANMsg31'=0);
endmodule

module Replica1
        // States of Payload
        PayS1 : [0..2] init 0;
        // 0 normal
        // 1 malicious
        // 2 down

        // States of Worm
        WormS1 : [0..1] init 0;
        // 0 normal
        // 1 down

        // States of Processing of Message 1 in replica 1
        ProcS11 : [0..4] init 0;
        // 0 idle
        // 1 vote
        // 2 wait vote
```

```
		// 3 wait forward
		// 4 forward

		// Management of Msg 1 in Replica 1
		Manage11 : [0..1] init 0;
		// 0 Replica 1 doesn't manage Msg 1
		// 1 Replica 1 is managing Msg 1

		// Voting Set
		Voting11 : [0..MaxVoteSet] init 0;
		// 0 Msg 1 not in Voting
		// 1 Msg 1 in Voting

		// Pending Set
		Pending11 : [0..1] init 0;
		// 0 Msg 1 not in Pending
		// 1 Msg 1 in Pending

		// TooEarly Set
		TooEarly11 : [0..1] init 0;
		// 0 Msg 1 not in TooEarly
		// 1 Msg 1 in TooEarly

		// Vote of Replica1 on Msg 1
		Vote11 : [0..1] init 0;
		// 0 Msg 1 not legal
		// 1 Msg 1 legal

		// Replica1 signs Msg 1
		Sign11 : [0..1] init 0;
		// 0 Msg 1 not yet signed
		// 1 Msg 1 signed

[] (PayS1=0) -> 1.0 : (PayS1'=1);
[] (PayS1=1) -> 2.0 : (PayS1'=0);
[] (PayS1=0) -> 3.0 : (PayS1'=1);
[] (PayS1=1) -> 4.0 : (PayS1'=2);
[] (PayS1=2) & (WormS1=0) -> 5.0 : (PayS1'=0);
[] (WormS1=0)-> 6.0 : (WormS1'=1) & (PayS1'=2);
[] (WormS1=1)-> 7.0 : (WormS1'=0) & (PayS1'=0);

		// Replica 1 receive from WAN message 1 too early
[] upProcess & (Buf11=1) & (Voting11<MaxVoted) & (Manage11=0) & (ProcS11=0) & (TooEarly11=1) -> 8.0 : (ProcS11'=0) & (TooEarly11'=0) &
(Buf11'=0);

		// Replica 1 receive from WAN message 1 in time
[] upProcess & (Buf11=1) & (Voting11<MaxVoted) & (Manage11=0) & (ProcS11=0) & (TooEarly11=0) -> 9.0 : (Vote11'=0) & (Manage11'=1) &
(ProcS11'=1) & (Buf11'=0);

		// Normal Behaviour
		// Replica 1 votes positive legal message 1  and sets message in Voting
[] (PayS1=0) & (ProcS11=1) & (Manage11=1) & (MsgT1=1) & (Voting11<MaxVoted) -> 10.0 : (ProcS11'=2) & (Vote11'=1) &
(Voting11'=Voting11+1);
		// Replica 1 votes negative not legal message 1
[] (PayS1=0) & (ProcS11=1) & (Manage11=1) & (MsgT1=0)  -> 11.0 : (ProcS11'=0) & (Vote11'=0) & (Manage11'=0);

		// Malicious Behaviour
		// Replica 1 votes negatives legal message 1
[] (PayS1=1) & (ProcS11=1) & (Manage11=1) & (MsgT1=1)  -> 12.0 : (ProcS11'=0) & (Vote11'=0) & (Manage11'=0);
		// Replica 1 votes positive not legal message 1  and sets message in Voting
[] (PayS1=1) & (ProcS11=1) & (Manage11=1) & (MsgT1=0) & (Voting11<MaxVoted)  -> 13.0 : (ProcS11'=2) & (Vote11'=1) &
(Voting11'=Voting11+1);

		// When at least f+1 replicas votes a message positive, sign it, delete from Voting and add to Pending
[] (majority) & (Voting11>0) & (ProcS11=2) -> 14.0 : (ProcS11'=3) & (Voting11'=Voting11-1) & (Sign11'=1) & (Pending11'=1);
		// else timeout
[] !(majority) & (Voting11>0) & (ProcS11=2) -> 15.0 : (ProcS11'=0) & (Vote11'=0) & (Voting11'=Voting11-1) & (Manage11'=0);
```

```
        // Wait random
[] upProcess & (ProcS11=3) -> 16.0 : (ProcS11'=4);

        // If message is still in Pending, send it to LAN
[] upProcess & (ProcS11=4) & (Pending11=1) -> 17.0 : (WaitingInLANMsg11'=1) & (WaitingInLANMsg21'=1) & (WaitingInLANMsg31'=1) &
(ProcS11'=0) & (Manage11'=0);
        // Else nothing to do
[] upProcess & (ProcS11=4) & (Pending11=0) -> 18.0 : (ProcS11'=0) & (Manage11'=0);

        // Replica 1 receive from LAN Msg 1
[] upProcess & (WaitingInLANMsg11=1) & (Pending11=1) -> 19.0 : (Pending11'=0) & (WaitingInLANMsg11'=0);
[] upProcess & (WaitingInLANMsg11=1) & (Pending11=0) & (MsgT1=1) -> 20.0 : (TooEarly11'=1) & (WaitingInLANMsg11'=0);
[] upProcess & (WaitingInLANMsg11=1) & (Pending11=0) & (MsgT1=0) -> 21.0 : (WaitingInLANMsg11'=0);

        // Replica 1 retransmits to WAN Msg 1
[] upProcess & (Voting11>0) -> 22.0 : (WaitingInWANMsg11'=1) & (WaitingInWANMsg21'=1) & (WaitingInWANMsg31'=1) & (Voting11'=0);

endmodule

module
Replica21=Replica1[ProcS11=ProcS21,Voting11=Voting21,Vote11=Vote21,Manage11=Manage21,Pending11=Pending21,TooEarly11=TooEarly21,
                    MsgS11=MsgS21]
 endmodule

module
Replica31=Replica1[ProcS11=ProcS31,Voting11=Voting31,Vote11=Vote31,Manage11=Manage31,Pending11=Pending31,TooEarly11=TooEarly31,
                    MsgS11=MsgS31]
Endmodule
```

4.1.3 Verification using Prism

In this section, we use the probabilistic model checking tool PRISM to automatically verify the satisfaction of integrity and correctness properties. Different models have been investigated: in all of them we consider processing of a single message that passes through a three replicas CIS system; even if the model allows to represent processing of multiple messages, there are no dependencies between them, therefore it's sufficient to check correctness and integrity properties for a single message. Processing of multiple messages can be useful, for example, if we want to investigate the order of reception of consecutive messages sent to the same destination.

Due to the size of the models, all experiments were performed with Prism's most space efficient model checking engine, which uses Multi-Terminal Binary Decision Diagrams (MTBDDs). We ran all experiments on a 1 GHz Pc with 1 GB memory under the Linux operating system with kernel 2.6. All properties were checked with an accuracy of $\varepsilon = 10^{-6}$. Two factors are variable:

- Type of messages received from the WAN by the CIS system: only legal messages, or legal and illegal ones.
- Replicas behaviour: during the processing all the replicas are and remain non malicious, or they transit from a normal to a malicious state.

	Only legal messages	Legal and malicious messages
All replicas are non malicious	21,273; 76,922	88,427; 396,364
Some replicas can became malicious	2,990,592; 27,465,136	5,962,048; 60,704,448

Table 4: Model Space State.

In Table 4 it is shown for each different combination of factors the number of states and the number of transitions of the resulting model. The model with 5,962,048 states took 53 iterations in 164.88 seconds to be constructed, however in case of multiple messages, for example two, a space state of about 10 ^{13} will be obtained.

For each model we check satisfaction of all the properties described in [Abou El Kalam *et al.* 2007], namely Lemmas 1 and 2:

1. *If f+1 correct replicas call approve function for the same message then it will be signed*.
2. *If a legal message is received by some correct replica and the message was not previously forwarded by other replicas, it will eventually be signed*.

From validity of these lemmas follows correctness and integrity:

1. *A legal message received by at least one replica is forwarded to its destination.*
2. *An illegal message is never processed by its destination*.

We verified that the CIS system satisfies these properties in all the models described in Table 4. More precisely we checked that the following CSL specifications hold in all states with probability 1:

"callapprove" => P>=1 [true U "signed"]
"correctsending" & !"forwarded "=> P>=1 [true U "signed"]
"correctsending" => P>=1 [true U "forwarded"]
"malicioussending" => P>=1 [true U "forwarded"]

where label *callapprove* is satisfied if at least one replica has variable *ProcS* equals to 2, i.e. thread has invoked function *approve*; label *signed* is satisfied if variable *Sign* equals to 1 and so on.

4.2 DEEM Models of the Proactive-Reactive Recovery Strategy

A quantitative analysis was performed on the Proactive-Reactive Recovery strategy proposed to reinforce the intrusion tolerance of the CIS in the scope of the protection service. The relevant characteristics of the system were modeled as Deterministic Stochastic Petri Nets (DSPN) [Mura *et al.* 2001] and solved using the DEEM tool [Bondavalli *et al.* 2000].

This deliverable contains the description of the models and some preliminary evaluation of the obtained results; considerations and implications of the analysis can be found in [Neves *et al.* 2008], together with some proposals about the refinement of the recovery strategy.

4.2.1 System Overview

CIS is a substation gateway interfacing a protected LAN with the WAN (as already shown in Figure 4-1) [Neves *et al.* 2008]. In order to be intrusion-tolerant, 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.

CIS intrusion tolerance is enhanced rejuvenating CIS replicas through recovery actions. In order to guarantee system availability despite the unavailability of recovering replicas, the maximum number of replicas allowed to recover in parallel is defined (k) and the number of replicas in the system is set to $n \geq 2f + k + 1$. This way, the system is able to tolerate (at most) f Byzantine replicas and recover k replicas simultaneously.

The CIS protection service, executed in each payload replica, verifies whether each incoming message *m* complies with the security policy (OrBAC[1]), notifying the (positive) approval to the local wormhole. As soon as a quorum of $f+1$ approvals for *m* is reached, the wormhole signs *m* and the current leader replica forwards it to the destination node in the LAN.

The wormhole is in charge of both triggering the recovery actions when necessary and managing the election of the new leader when necessary.

4.2.2 The Proactive-Reactive Recovery Strategy

The PRRW strategy [Neves *et al.* 2008] manages the CIS replica recoveries using a mix of proactive and reactive recovery actions. Proactive recoveries are performed based on periodic base, whilst reactive recoveries are performed on replicas “suspected” or “detected” to be compromised. The leader replica is “suspected” to be compromised when it does not forward all the signed messages; a generic replica is “detected” to be compromised when it sends not signed (invalid) messages to the LAN. Accusations about a replica being “suspected” or “detected” can be raised by each payload replica and sent to the corresponding local wormhole through a specific interface.

The PRRW strategy is organized as follows: time is divided in $\lceil n/k \rceil$ different time slots that are cyclically repeated. Each slot is divided in two tasks: task *A* and task R_i with $i = 1, \ldots, \lceil n/k \rceil$.

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

Reactive (a-periodic) recoveries are executed in task *A* only. Task *A* is divided into $\lceil f/k \rceil$ recovery sub-slots (identified as S_{ij}); up to k replicas can be recovered in each sub-slot. Task *A* lasts for (at most) $\lceil f/k \rceil T_D$. Reactive recoveries are triggered on replica *i* in the following scenarios:

1. replica *i* is “detected” to be compromised when the wormhole collected a quorum of $f+1$ accusations about *i* sending illegal messages,
2. replica *i*, being the current leader, is “suspected” of being compromised when the wormhole collected a quorum of (at least) $f+1$ accusations, some about *i* sending illegal messages, other about *i* not forwarding signed messages.

Reactive recoveries are immediately performed in the scenario 1 (at least one correct replica detected that replica *i* is failed), whilst are coordinated[2] with the periodic recoveries in the scenario 2. The quorum of $f+1$ collected accusations is needed to avoid recoveries triggered by faulty replicas: at least one correct replica must detect/suspect replica *i* for some recovery action to be taken.

A new leader is elected by the wormhole if the current leader is recovering (e.g. because it was suspected of being omissive) or if the local wormhole of the current leader is detected to

[1] OrBAC (Organization Based Access Control) [Abou El Kalam *et al.* 2003]

[2] If no reactive recovery is already scheduled for replica *i*, the PRRW strategy finds the closest recovery sub-slot where recovering replica *i* such that the availability of the CIS is not endangered. If the found sub-slot is located in the slot where replica *i* will be proactively recovered, the reactive recovery is not performed

be crashed. The new leader is chosen as the (currently not crashed) replica more recently recovered by a proactive recovery.

4.2.3 Fault Model and Assumptions

This section describes the fault model [Sousa *et al.* 2007] and the assumptions on which the fault model is based on. 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). The following faults are considered:

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

f2) A payload replica can be intruded, and hence can be affected by Byzantine faults; if more than f payload replicas fail, the CIS fails.

f3) A local wormhole can only fail by crash; at most $f_c \leq f$ 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).

f5) The same attack on the same replica has always the same probability of success.

f6) Station computers cannot be 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).

Given the set of faults just described, the corresponding failure modes for a payload replica are:

- 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 replica payload is subjected to a temporary omission because of communication problems (f1) or as the effect of an intrusion (f2) (e.g. the leader payload is omitting to forward a signed message to its destination or the net is flooded with messages by someone else).
- 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 LAN networks with illegal messages aiming to delay the forwarding of legal messages.

The system fails when the number of invalid replicas is greater than f (the correctness of the system cannot be guaranteed) or when the necessary resources are not available for too long (the CIS seeks perpetual operation); the system is unavailable when the number of correct working replicas is less then $f+1$ (so quorums cannot be reached) or there are more than $f+1$ correct replicas, but the leader is omitting (so legal messages are not forwarded).

4.2.4 PRRW Modeling

The relevant measures of interest for the recovery strategy under study are the system failure probability and the system unavailability.

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

1. the number of invalid replicas gets over f ;
2. the system is unavailable since t-T_O, that is the system is unavailable for a period of time grater 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, denoted by $P_F(t)$, 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 obtained as

$$P_F(t) = P_{FI}(t) + P_{FO}(t)$$

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

1. the number of correct replicas is less then $f + 1$ (quorums cannot be reached)
2. there are more than $f + 1$ correct replicas, but the leader is omitting (legal messages are not forwarded).

Let $T_U(0,t)$ be the total time the system is not failed but is unavailable within $[0,t]$ because of one of the above conditions. Let $T_A(0,t)$ be the total time the system is not failed within $[0,t]$. System unavailability, denoted by $P_U(t)$, is defined as the probability of the system being unavailable at time t, given that it was correctly working at time $t = 0$; system unavailability is obtained as

$$P_U(t) = \frac{T_U(0,t)}{T_A(0,t)}$$

For ease of modelling, we assume that a replica, as soon as it is successfully intruded, explicitly manifest failures (of any kind) and that a failure caused by an intrusion is permanent.

From the modeling point of view, a reconfiguration strategy determines 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 makes a reconfiguration strategy belonging to the Multiple Phased System (MPS) class for which a modeling and evaluation methodology exist [Mura *et al.* 2001], supported by the DEEM tool [Bondavalli *et al.* 2000].

Using DEEM, the net 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 and simulation solutions can be used in order to exercise the models; the measures of interest defined in our quantitative analysis were evaluated by simulation.

4.2.4.1 Phase Net

The phase net of the PRRW model (Figure 4-4) triggers the recoveries. The deterministic transitions *TsubSlot* and *TRi* model the times to perform the task *A* and R_i, respectively. The place *Sij* contains a token during task *A* (a-periodic recovery phase) and *Ri* contains a token during the R_i task (periodic recovery phase). The marking of *CountSubSlot* counts the number of the current recovery sub-slot (*Sij*) within the current recovery slot. The marking of place *CountSlot* counts the number of the current recovery slot within the current cycle. The marking of place *CountWin* counts the number of the current cycle. The immediate transition *tNextWin* fires when a new cycle is started, resetting the marking of *CountSlot* to 1.

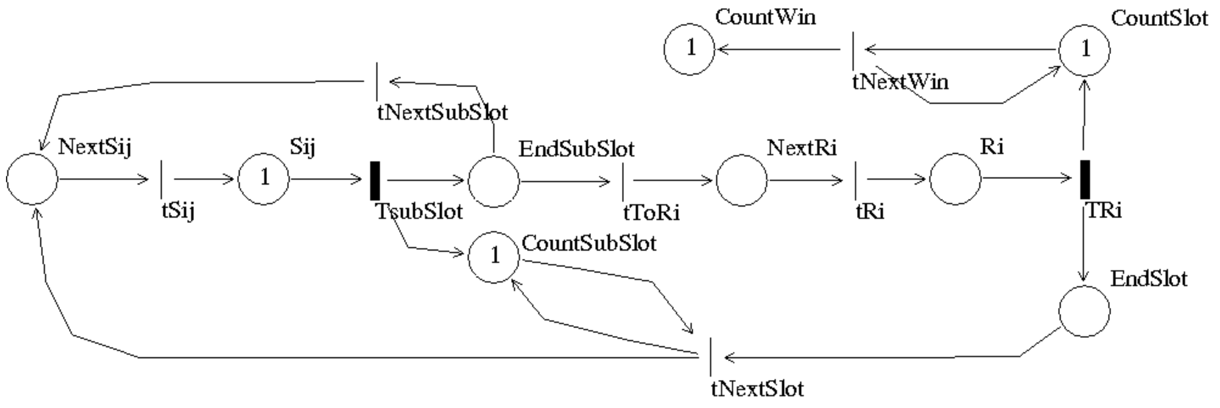


Figure 4-4: The phase net of the PRRW model

4.2.4.2 System Net

The system net of the PRRW model is composed by n similar subnets ($n \leq 6$), one subnet for each replica, a subnet to take track of system failures and a subnet to model the initialization (the description of this last subnet is omitted without affecting the comprehension of the model).

Figure 4-5 shows the subnet modelling replica *1*. The left part of the subnet models the replica failures, whilst the right part of the subnet models the management of both replica recovery and leader election. Places which name ends with digit "1" model replica *1*, whilst the other places (*Leader* and *kRec*) are shared by all the replicas.

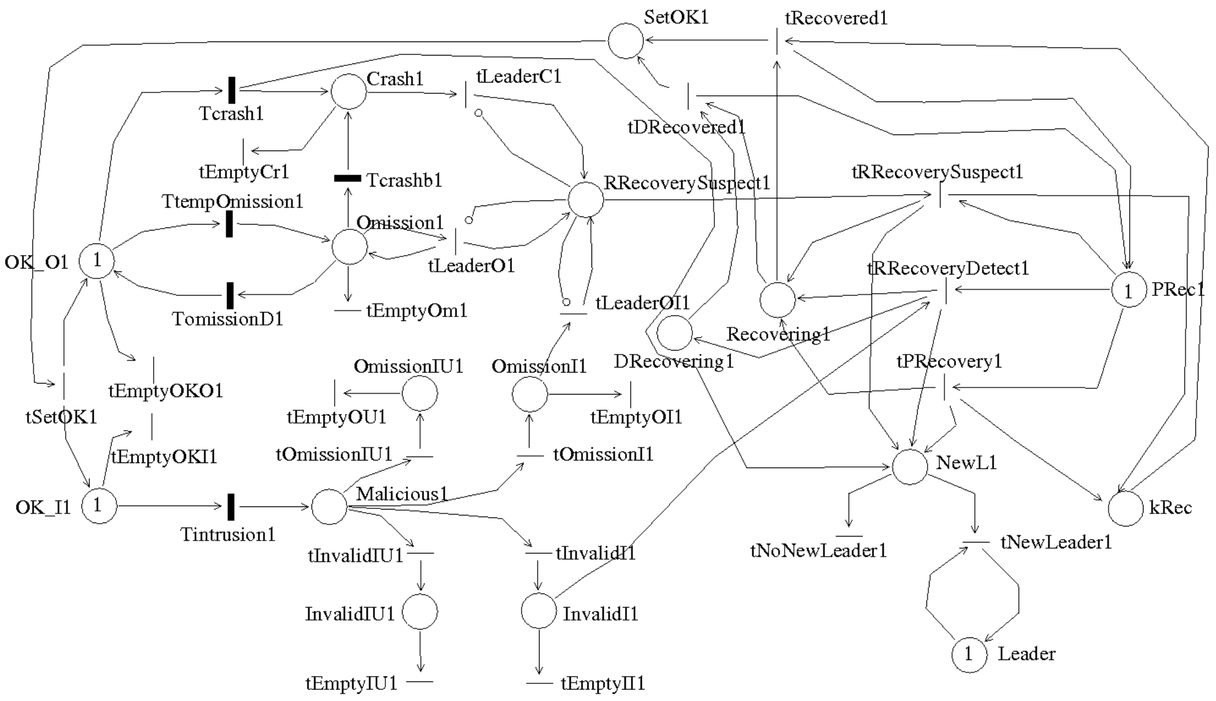


Figure 4-5: The subnet modeling replica 1 in the PRRW model

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* represent the crash of the replica or an omissive behaviour as a consequence of transient omission respectively. The exponential transition *Tcrash1* represents the time to occurrence of a crash, exponentially distributed with rate λ_{C1}. Each replica has its own crash rate, varying from a crash each 60 days (replica 1) to a crash each 30 days (replica 6). The crash of a replica is independent of the crash of other replicas. When the replica crashes, place *OK_I1* is emptied (the replica cannot be no more intruded). *TtempOmission1* represents the time to a transient omission, exponentially distributed with rate λ_{O1}. Each replica has its own transient omission rate, varying from an omission each 6 days (replica 1) to an omission each 3 days (replica 6). A transient omission becomes permanent or disappears after a time modeled by the exponential transition *Tcrashb1* with rate λ_{C1} or *TomissionD1* λ_{EO} with rate respectively.

The exponential transition *Tintrusion1* represents the time to a successful intrusion with rate λ_{A1}. Each replica has its own rate, varying form an intrusion each 5 hours (replica 1) to an intrusion each 24 hours (replica 6). The immediate transitions *tOmissionIU1*, *tOmissionI1, tInvalidIU1*, *tInvalidI1* and the associated output places model the effect of the intrusion in terms of undetectable omission failure (with probability $(1-c_M)\cdot(1-p_I)$), detectable omission failure (with probability $c_M \cdot (1-p_I)$), undetectable invalid failure (with probability $(1-c_M)\cdot p_I$) or detectable invalid failure (with probability $c_M \cdot p_I$) respectively.

The management of the replica recovery is modeled as follows. Place *PRec1* contains a token as long as replica 1 is not recovering. whilst place *Recovering1* contains one token as long as the replica is recovering. Place *DRecovering1* contains a token during a reactive recovery triggered by detections. Place *kRec* is used to count the number of replica currently recovering. Place *RRecoverySuspect1* contains a token if a crash, an omission or a malicious omission are occurred.

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

After the starting of a recovery of the replica, all the immediate transitions whose name starts with *tEmpty* fire, emptying the following places: *OK_O1*, *OK_I1*, *Crash1*, *Omission1*, *OmissionIU1*, *OmissionI1*, *InvalidIU1*, *InvalidI1*. Immediate transitions *tRecovered1* or *tDRecovered1* fires when the current recovery ends, resetting the replica subnet.

The election of the leader replica is managed as follows. The marking of the place *Leader* corresponds to the identity of the current leader; when the replica 1 either is going to be recovered or is crashed, one token is added in the place *NewL1*. *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)*, whilst the arc from place *tNewLeader1* to place *Leader* has multiplicity equal to the identity 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 identity $j=((n+\left(Mark(CountSlot-2)\cdot k\right)\bmod n)+k$. If replica j is currently crashed, the next attempt is made on replica $j-1$, until a non crashed replica is found.

The subnet shown in Figure 4-6 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). The place *OKSysO* contains a token when the system is not failed but it is omitting. The place *SysFailureI* contains a token when the system is failed because of invalid behaviour (there are at least $f+1$ invalid replicas). The place *SysFailureO* contains a token when the system is failed because the resource unavailability[3] lasted for an unacceptable period of time represented by the exponential transition *TSysO* with rate $1/T_O$.

[3] Either the current leader was omitting or there were less than $f+1$ correct replicas

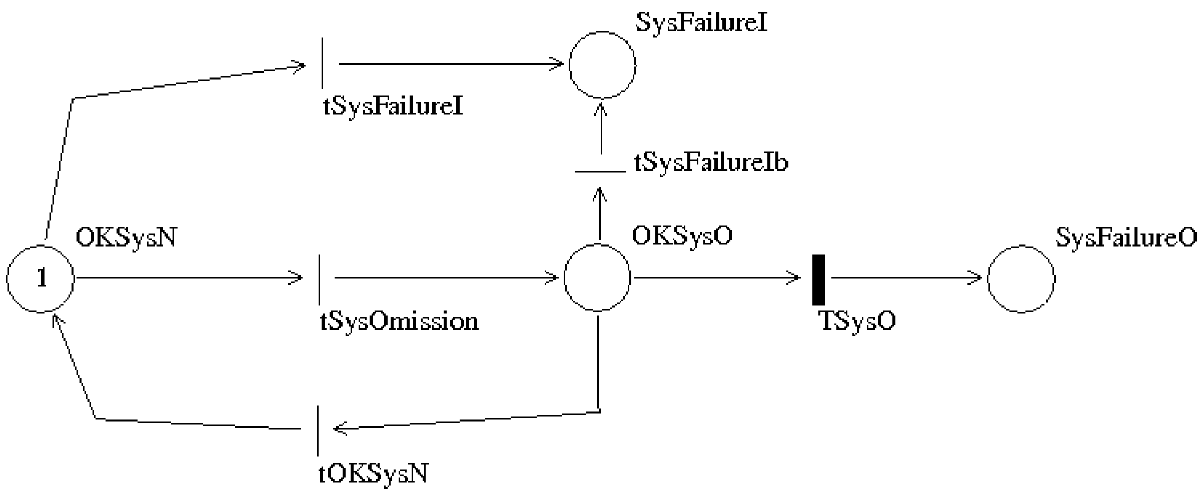


Figure 4-6: The model of system failure in the SN of the PRRW model

To improve the efficiency of the analytical solution different priorities are associated to the immediate transitions of SN, when no probabilistic choices are required. For example, all the immediate transitions of the replica i have priorities lower than those of replica j, if $i < j$.

4.2.4.3 Reward Structures

The evaluation of the measures of interest $P_F(t)$ and $P_U(t)$ 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(OKSysO)=0 or Mark(OKSysN)=0) then (1) else (0)
```

System unavailability $P_U(t)$ was evaluated as $P_U(t) = \frac{T_U(0,t)}{T_A(0,t)}$.

$T_U(0,t)$ was evaluated defining an 'interval of time" performance variable whose 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)
```

4.2.5 Model Evaluation and System Analysis

In this section the results of the evaluation of the measures of interest is shown. The measures of interest were evaluated by simulation with a confidence level of 95% and a half-length confidence interval of 1%.

All the model parameters and the basic values used for the evaluations are shown in Table 5. The relevant parameters used are the following:

- Mission time t. This is the time during which the system is exercised since it starts to work. t varies in $[2628,42048]$.

- Detection coverage c_M of malicious behaviour. c_M is the probability of detecting an intruded replica, and hence the probability of reactively recovery an intruded replica. c_M varies in $[0,1]$: if $c_M = 0$, no intrusions are detected, whilst if $c_M = 1$, all intrusions are detected. $c_M = 1$ in the ideal case.
- Probability p_I of intrusion manifesting as a permanent invalid behaviour. p_I varies in $[0,1]$: if $p_I = 0$, all intrusions manifest as a permanent omissive behaviour; if $p_I = 1$, all intrusions manifest as a permanent invalid behaviour.
- 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.

Table 5: Model Parameters and Default Values

Parameter	Value	Meaning
t	2628	Mission time (sec)
n	4	Number of replicas in the system
k	1	Max number of replicas recovering simultaneously
f	1	Max 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_{Ci}	$[1.9e^{-7}, 3.8e^{-7}]$	Crash rate, exponentially distributed. Each replica has a diverse crash rate (from 1 per 60 days to 1 per 30 days).
λ_{Oi}	$[1.9e^{-6}, 3.8e^{-6}]$	Transient omission rate, exponentially distributed. Each replica has a diverse rate (from 1 per 6 days to 1 per 3 days)
λ_{EO}	$3.3e^{-2}$	Omission duration rate, exponentially distributed. A transient omission lasts about for 30 seconds.
λ_{Ai}	$[5.8e^{-5}, 1.2e^{-5}]$	Successful attack (intrusion) rate, exponentially distributed. Each replica has a diverse rate (from 5 per day to 1 per day)
p_I	0.5	Probability of intrusion manifesting as a permanent invalid behaviour (if $p_I = 0$ all intrusions manifest as permanent omissions)
c_M	0.7	Probability of detecting a malicious behaviour

The relevant results of the preliminary evaluations performed on the model are briefly shown hereafter. More details are given in the appropriate sections of [Neves *et al.* 2008].

A first study was performed observing both system failure probability $P_F(t)$ and system unavailability $P_U(t)$ over mission time t. Three system configurations were evaluated for three different values of p_I (probability of intrusions manifesting as permanent invalid behavior).

Figure 4-7 shows how $P_{FI}(t)$ and $P_{FO}(t)$ change over mission time t (recall that $P_F(t) = P_{FI}(t) + P_{FO}(t)$). If $p_I = 0$, $P_F(t)$ is constant as time increases (in this case $P_F(t) = P_{FO}(t)$, given that $P_{FI}(t) = 0$). If $p_I = 0.5$, $P_F(t)$ increases as time increases; the figure shows that $P_{FO}(t)$ is negligible with regard to $P_{FI}(t)$ ($P_{FO}(t)$ decreases to 0 as time increases). If $p_I = 1$, $P_F(t)$ increases as time increases; the figure shows that $P_{FO}(t)$ is constant over time and that $P_{FI}(t)$ increases over time. The system with $p_I = 0$ has the smallest values for $P_F(t)$ among the three system considered, whilst the system with $p_I = 1$ has the largest values for $P_F(t)$ (at least two orders of magnitude with regard to the system with $p_I = 0$).

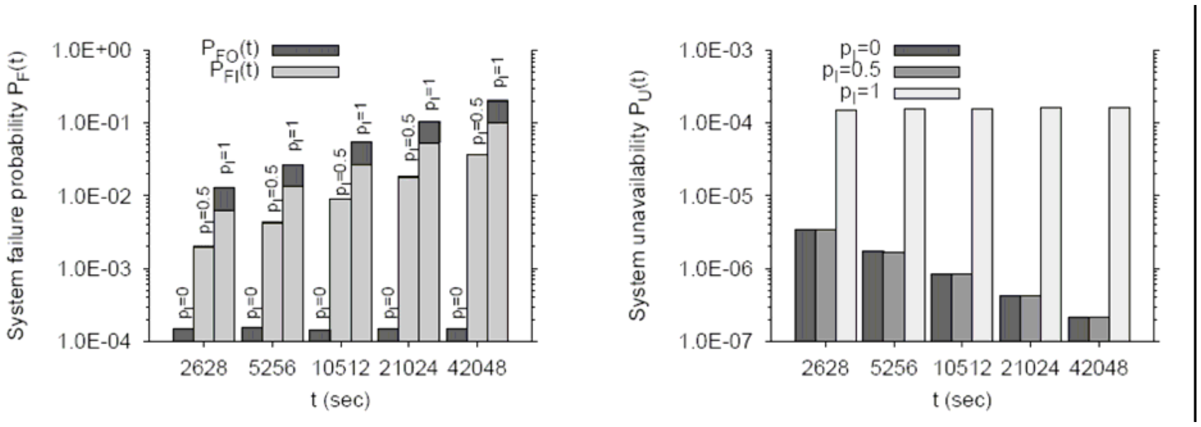


Figure 4-7: System failure probability $P_F(t)$ and unavailability $P_U(t)$ over mission time t for different values of p_I

If $p_I = 0$, there isn't any invalid behaviour triggering reactive recoveries, so almost all recoveries are periodic (reactive recoveries are triggered only on an omissive leader): in this case $P_F(t)$ shows to be almost constant over time. If $p_I \in \{0.5,1\}$, there are invalid behaviours triggering reactive recoveries: in both the cases $P_F(t)$ shows to be increasing overtime mainly because of $P_{FI}(t)$; the larger is p_I, the larger is $P_F(t)$. In general it turns out that $P_F(t)$ increases over time mainly because of the reactive recoveries triggered by invalid behaviour.

Figure 4-7 shows also how $P_U(t)$ changes over mission time t. If $p_I \in \{0,0.5\}$, $P_U(t)$ decreases as time increases (both the settings show the same values for the same t). If $p_I = 1$, $P_U(t)$ shows to be constant with regard to time; in this case $P_U(t)$ is larger than for the other values of p_I.

$P_U(t)$ is decreasing over time (at most is constant), so it is satisfactory in all the system configurations considered; the positive effect of proactive recoveries refreshing all the replicas is evident mostly for $p_I = 0$, whereas the effect is smaller (but still effective) as p_I increases, since $P_U(t)$ is not increasing over time.

Another study was devoted to evaluate both system failure probability $P_F(t)$ and system unavailability $P_U(t)$ varying both the detection coverage c_M and the probability p_I of intrusion manifesting as invalid behaviour. Varying the value of c_M we act on the mechanism of reactive recoveries in the following way: increasing the detection coverage, it increases also the number of invalid replicas detected to be invalid and hence reactively recovered. If $c_M = 0$, reactive recoveries are triggered only on an omissive leader.

Figure 4-8 shows how $P_{FI}(t)$ and $P_{FO}(t)$ change over detection coverage c_M for some values of p_I. The two figures are plotted using the same scale for the y-axis in order to make easier their comparison. In general, $P_{FI}(t)$ decreases as c_M increases from 0 to 1. The extreme curves correspond to the two extreme system configurations: $p_I = 1$, assuming the largest values, and $p_I = 0$, assuming the lowest values: in this last case $P_{FI}(t) = 0$ (the curve is out of the bounds of the figure). The curve showed in Figure 4-9 which assumes the smallest values corresponds to the system configuration where $p_I = 0.2$. The curve corresponding to $p_I = 1$ decreases quicker than the other curves (it decreases for about one order of magnitude), whilst the curve for $p_I = 0.2$ is almost constant. $P_{FO}(t)$ shows an opposite behaviour with respect to $P_{FI}(t)$: it increases as c_M increases from 0 to 1. The extreme curves correspond to the two extreme system configurations: $p_I = 0$, assuming the largest values, and $p_I = 1$, assuming the lowest values (the opposite behaviour with respect to $P_{FI}(t)$). The curve corresponding to $p_I = 1$ increases quicker than the other curves, whilst the curve for $p_I = 0$ is almost constant.

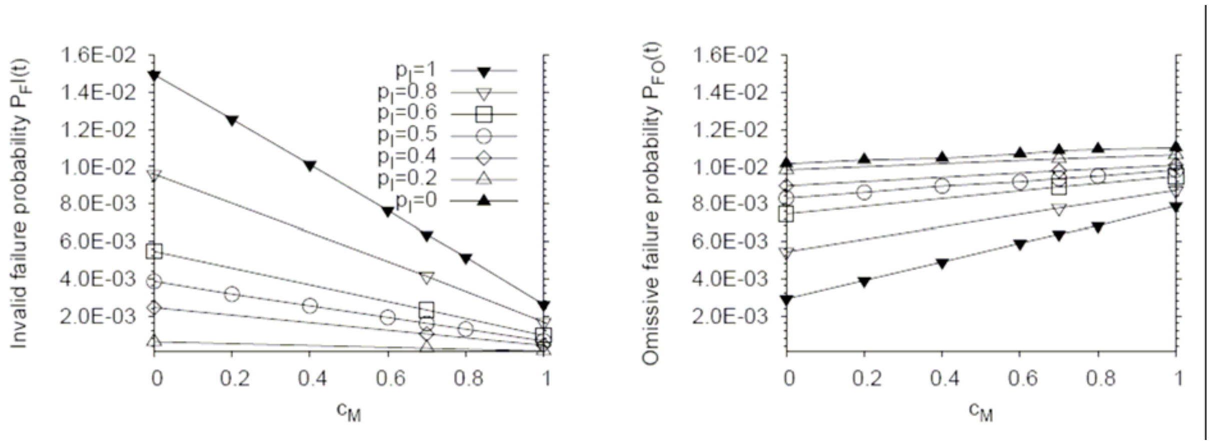


Figure 4-8: Impact of detection coverage c_M on failure probability $P_{FI}(t)$ due to invalid behaviour and failure probability $P_{FO}(t)$ due to omissive behaviour for different values of p_I

Figure 4-9 shows that increasing c_M there are two opposite effects with respect to $P_{FI}(t)$ and $P_{FO}(t)$: the former decreases, because invalid replicas reactively recovered are no longer weakening the system; the latter increases, because replicas, while recovering, do not contribute to system operation. The overall effect, shown in Figure 4-9, is that system failure probability decreases as detection coverage c_M increases. The system was evaluated in this study for $t = 2628$; the first study described above showed that system failure probability $P_F(t)$ increases over time (except for system configuration where $p_I = 0$). We hence suppose that this study, if evaluated for a larger value of t, should show a larger difference between the extreme system configurations $p_I = 0$ and $p_I = 1$. This stresses that the value for c_M should be as higher as possible.

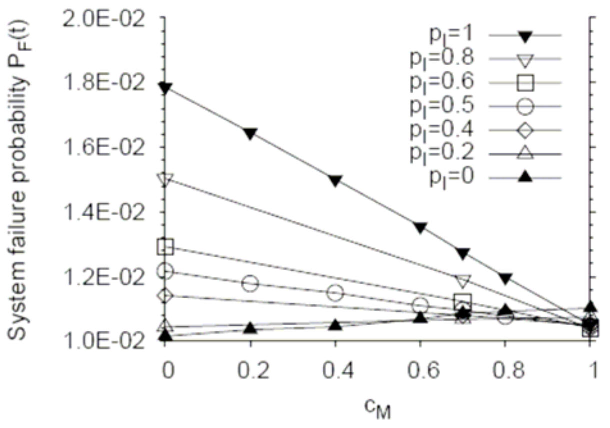


Figure 4-9: Impact of detection coverage c_M both on system failure probability $P_F(t)$ and omissive failure probability $P_{FO}(t)$ for different values of p_I

Figure 4-10 shows how system unavailability $P_U(t)$ changes over detection coverage c_M for some values of p_I. In general, $P_U(t)$ increases as c_M increases from 0 to 1. The extreme curves correspond to the two extreme system configurations: $p_I = 0$, assuming the largest values, and $p_I = 1$, assuming the lowest values. If $p_I = 0$, $P_U(t)$ is almost not influenced by changing the detection coverage, whilst increasing p_I the influence of c_M becomes more evident (almost an order of magnitude for $p_I = 1$).

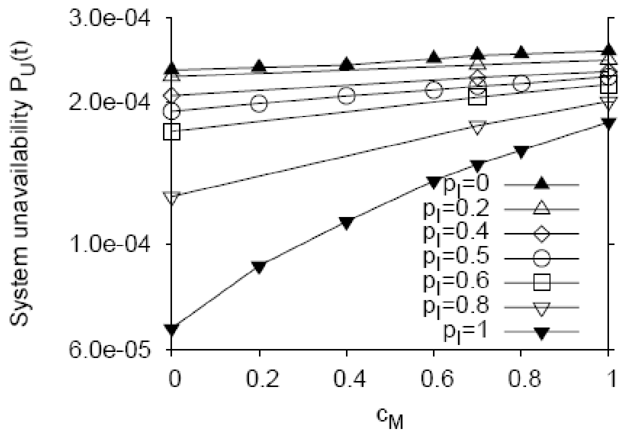


Figure 4-10: Impact of detection coverage c_M on system unavailability $P_U(t)$ for different values of p_I

It turns out that there are two conflicting effects on system unavailability related to the increasing number of reactive recoveries. $P_U(t)$ is negatively affected by a larger value for the detection coverage c_M, because the larger is the detection coverage, the more reactive recoveries are triggered; the above trend is more evident as the probability p_I of intrusion manifesting as invalid behaviour increases, because reactive recoveries are mainly triggered because of invalid behaviour. On the other side, it has to be noticed that the smaller is p_I, the worse is system unavailability, because less reactive recoveries are triggered on faulty replicas.

The results of this study shows that increasing the detection coverage of invalid behaviour has conflicting effects on system failure probability and system availability: the former improves as c_M increases, whilst the latter worsen as c_M increases.

The last study performed aimed to evaluate the impact of the number of replicas on both system failure probability and unavailability. 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. The values of the above parameters are disciplined by the following formula: $n = 2f + 1 + k$. The following system configurations were evaluated:

1. $n = 4$, $f = 1$, $k = 1$
2. $n = 5$, $f = 1$, $k = 2$
3. $n = 6$, $f = 1$, $k = 3$
4. $n = 6$, $f = 2$, $k = 1$

Figure 4-11 shows system failure probability $P_F(t)$ (decomposed in $P_{FI}(t)$ and $P_{FO}(t)$) and system unavailability $P_U(t)$ for the system configurations described above. In general $P_F(t)$ decreases as the number n of system replicas increases (this trend is due primarily to $P_{FO}(t)$); for the same value of n, the higher is k and the lower is $P_F(t)$. In general $P_U(t)$ decreases as the number n of system replicas increases; for the same value of n, the higher is k and the lower is $P_U(t)$. It turns out that for the setting used (as shown in Table 5) the lower system failure probability and the lower system unavailability is measured in the system configuration 3; this configurations has the highest number n of replicas with regard to the other configurations evaluated. Configuration 4 has also the same n as configuration 3, but it has a lower value for k ($P_{FO}(t)$ is the main contributor to $P_F(t)$).

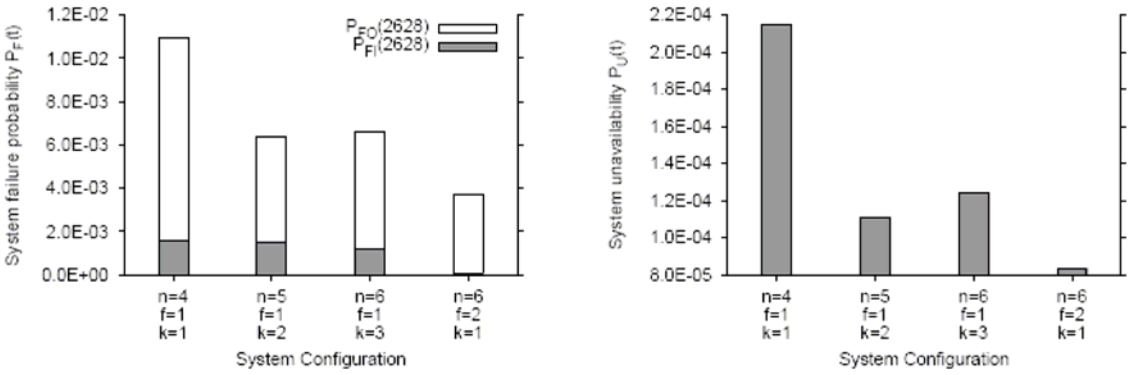


Figure 4-11: System failure probability $P_F(t)$ and unavailability $P_U(t)$ for different system configurations at mission time $t = 2628$

5 PRELIMINARY ASSESSMENT OF DEPENDENCIES BETWEEN THE EI AND II INFRASTRUCTURES

Initial analyses to quantitatively assess the impact of interdependencies between the two infrastructures EI (electrical infrastructure) and II (information infrastructure) composing the Electric Power System (EPS) have been performed through the EPSyS simulator described in Deliverables D11 [Donatelli *et al.* 2008b] and D8 [Kaâniche *et al.* 2008].

5.1.1 Scenario, settings, fault conditions and analyzed measure.

The first analyses conducted through the EPSyS simulator developed at CNR-ISTI have been based on an artificial, simple EPS testbed, properly built to address specific features of the simulator. Our initial testbed is composed of 24 nodes, subdivided in 8 power plants (rounds) and 16 loads (squares). The loads can also act as substations if needed. The nodes are interconnected using 27 identical transmission lines, as shown in ***Figure 5-1***.

This testbed can be logically partitioned in three main areas:

- The “B” area has a power demand much higher with respect to its production.
- The “A” and “C” areas, on the other hand, have a power production much higher than their demand.
- At the equilibrium, a constant power flow is expected to be transferred using the transmission lines that are drawn thicker.

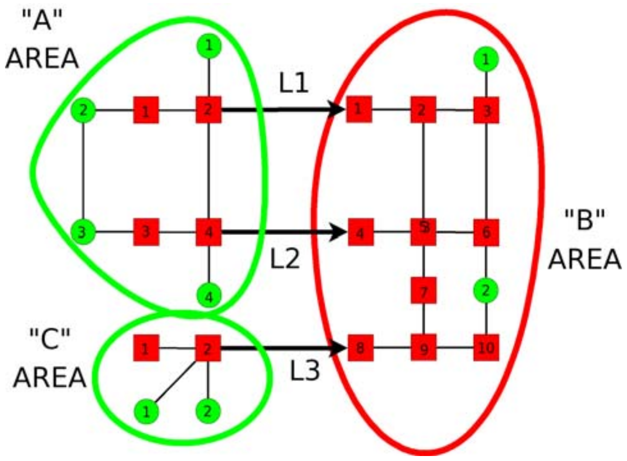


Figure 5-1: Overview of the testbed system

The settings for the nodes and the lines used in this testbed are here summarized.

For the whole grid:

- Every transmission line can tolerate a maximum of 620 MW.
- All transmission lines are treated equally: there is no preferred path for power flow or redispatch of power flow.
- Environmental temperature is assumed to be constant and uniform.
- Each load provides identical, constant gain if its power demand is satisfied, or a constant cost otherwise.
- Plants and lines are neutral from the point of view of gains/costs (no cost if overloaded or stressed, but also no gain in the opposite situation).

For the “A” area:

- Each plant in the area can provide a maximum power of 450 MW.
- Each load in the area demands 100 MW.
- The overall balance of the area shows that there is a theoretical excess of production of 1400 MW.

For the “B” area:

- The plant #1 can produce a maximum of 200 MW; the plant #2 can produce a maximum of 300 MW.
- Loads in the area absorb 200 MW each one, with the exception of loads #3, #6 and #10 that require 100 MW only.
- The overall balance of the area shows that there is an unsatisfied demand of 1200 MW

For the “C” area:

- Both plants can produce a maximum of 600 MW.
- Both loads require 100 MW.
- The overall balance of the area shows a theoretical excess of production of 1000 MW.

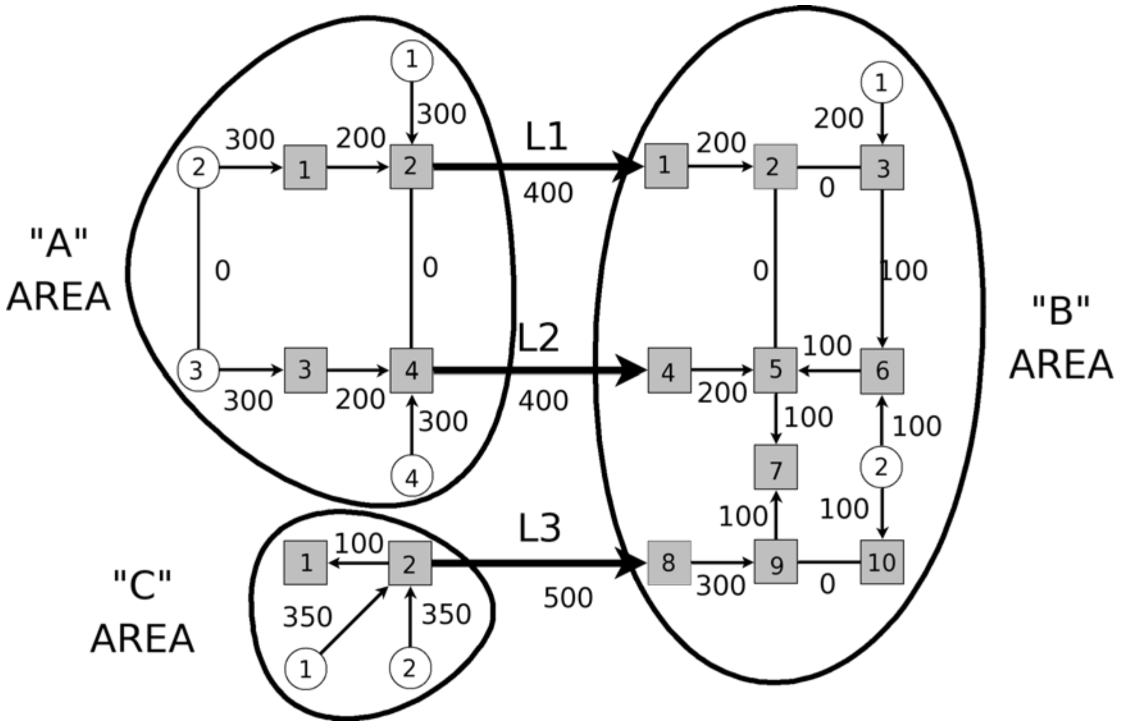


Figure 5-2: Initial power flow distribution for the selected scenario

From the parameters of the grid nodes in the testbed, there is more theoretical production than demand, thus allowing a variety of dispatch decisions. We have chosen an initial stable state summarized in Figure 5-2.

The measure we evaluate is the expected reward $E[Y_{[0,t]}]$ accumulated in the interval [0, t], with Y[0,t] defined as (see section 2.1 above for details):

$$Y_{[0,t]} = \sum_{P \in RS} R(P) J_{[0.t]}$$

We considered a 24 hours timeframe (so t = 24 hours), and we normalize the obtained value with respect to t. Therefore, a value near to 1.0 means good performance and high load satisfaction (so, high gain), while a value near to 0.0 means heavy blackout and high costs.

We consider a starting stable state and we introduce two simultaneous faults, which instantaneously cause outages of the lines L1 and L2, after 1 hour from the starting of the simulation. Faults are assumed permanent, so the affected lines are permanently out of service, and a repair action is required to bring them back in service.

We evaluate the behaviour of the system considering that the global II reaction function $RS_2()$ requires a time varying from 30 seconds to 15 minutes, with a 30 seconds step. Varying this time, we can model different degrees of promptness to the II infrastructure, ranging from almost instantaneous reaction to longer time representing timing failures experienced by $RS_2()$.

We also consider different stress factors for the system, assuming a maximum of 50% overload with an increasing step of 10%. The system stress is represented by the *alpha* factor: alpha=1 indicates the nominal power flow on the lines to satisfy a given set of loads under an equivalent power production by a set of generators. Increasing values of alpha indicate that a higher load is requested in correspondence of a higher production, but with the same power lines topology. This means that power lines become more stressed, and the chance to experience lines overloads increases. In this study, we vary alpha in the interval [1.0, 1.5], with step = 0.1.

5.1.2 Quantitative analyses

In the assumed fault scenario we observe that, just after the fault, $RS_1()$ tries to fullfill the demand of the “B” area by increasing the production of generators in the “C” area. The power flow is routed through L3, being the last surviving interconnection line. L3, being overloaded, starts to heat. It is going to be disconnected in a few minutes (the precise amount of time depends on the overload system stress factor alpha).

There are two possible evolution directions. If $RS_2()$ is too slow to react, L3 will be lost as well, leading to the most severe blackout possible. The cost of this blackout depends in turn on the value of the stress factor imposed to the system: the higher is the stress factor, the heavier will be the blackout cost. On the other hand, even if $RS_2()$ is quick enough, a blackout is experienced, since the power flow that can be delivered to the “B” area is bounded by the capacity limits of L3 (since $RS_2()$ always avoids overload on transmission lines). Anyway, this is the less severe blackout possible, so the best possible performance without the repair of the broken lines.

Figure 5-3 shows the plot of the considered performability measure under varying values of the time $RS_2()$ requires to determine and apply the new stable electric configuration (named $RS_2()$ delay in the following), and for different values of the stress factor alpha. The interesting points are the sudden drops for the $RS_2()$ delay between 12 and 13 minutes for alpha=1.1, and for a delay between 4 and 5 minutes for alpha=1.2. These drops are caused by the loss of L3 due to an overload. After that, the system is compromised, in the sense that the most severe blackout cannot be avoided.

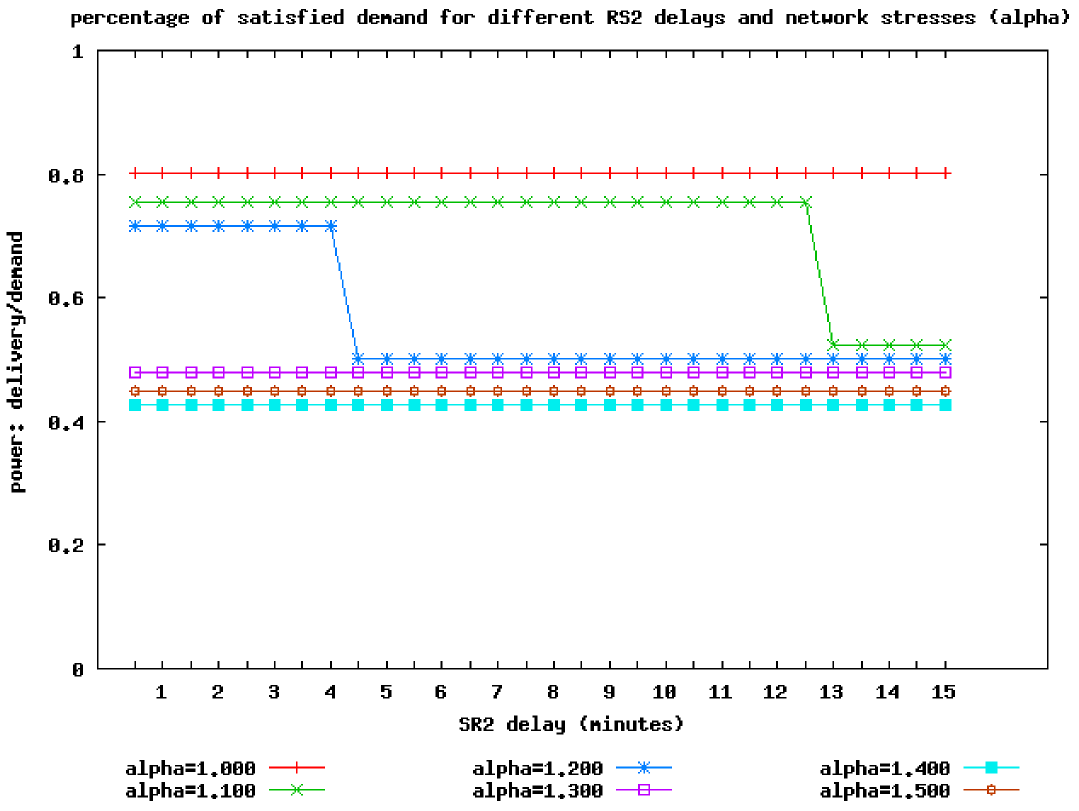


Figure 5-3: Mean normalised delivered power in the analysed scenario

It must be noted as well that the loss of L1 and L2 leads to a partition of the grid in two islands: the “A” area and the “B+C” area. The island corresponding to the “A” area isn’t affected by any blackout since there is enough available production. Therefore, the loads in this subgrid are fully satisfied.

The further loss of L3 subdivides the grid in three areas. In addition to the "A" area, the considerations above hold for the "C" area as well: it is the "B" area the only one that experiences the blackout.

We inspect now in detail the behaviour of the grid elements with respect to the sustained power flow during the simulation timeframe. We consider first Figure 5-4 which shows the evolution of power flows through transmission lines for the $RS_2()$ delay = 4:00 m and alpha = 1.2.

On the X-axis, we have the transmission line sequential number, while on the Y-axis we have the simulation time in minutes:seconds. Transmission lines are numbered sequentially considering the A-B-C area's order, and last the areas interconnection lines L1-L2-L3 (lines numbered 25,26 and 27 respectively in Figure 5-4). The intensity of the colour for each bar representing a line is proportional to the line load: the brighter is the colour, the heavier is the load. A load greater than 1.0 indicates an overload.

This Figure depicts the maximum allowed delay for $RS_2()$ to be still able to alleviate the L3 overload, thus preventing its disconnection from the grid. Starting from time t=60:00m, we can see a significant overload on L3 caused by the increased power generation and power redispatch ordered by $RS_1()$ in response to the loss of L1 and L2. After 4:00 minutes (and a few more seconds due to propagation delays), $RS_2()$ acts by reducing the flow on L3, and the production of generators in the C area, thus avoiding the tripping of L3. Unfortunately, this will also lead to a significant blackout in the B area, due to the lack of available power

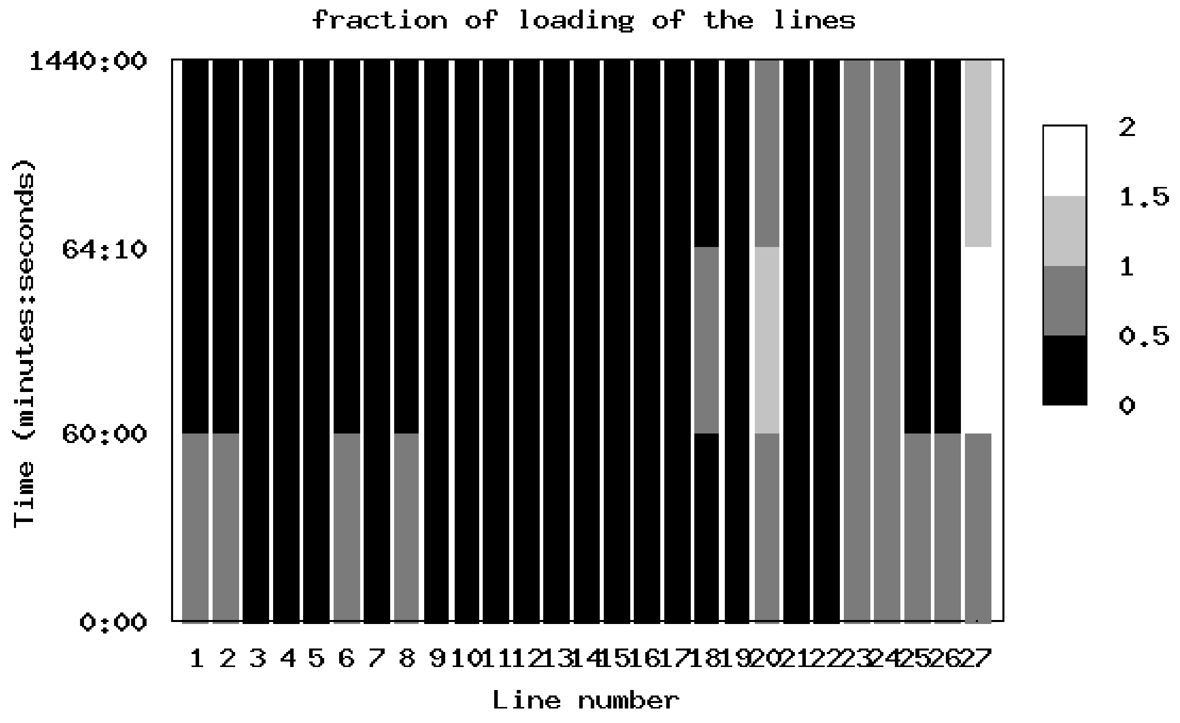


Figure 5-4: Evolution of the load through power lines for a $RS_2()$ delay=4:00 m

Figure 5-5 shows the evolution of power flows through transmission lines for the $RS_2()$ delay equal to 4:30 m and alpha equal to 1.2. This Figure represents the minimum $RS_2()$ delay that causes the tripping of L3 from the grid. We can see that L3 trips at time 64:19m (4:19 m after the overload started). The tripping is evident because line L3 passes from a high overload to a zero load. This pattern is caused by the intervention of the protections embedded in the line. This event causes $RS_2()$ to be interrupted and restarted, and this is evident because the next changes in the power through the lines, caused by an $RS_2()$ action, happens 68:49 m after the fault. Therefore, in this case the duration of $RS_2()$ is 8:49 m, since after 4:19 m it is restarted by the tripping of L3, and then completes after the expected 4:30 m after its restart.

In this scenario, a severe blackout is experienced: given the out of service of the power lines L1, L2 and L3, the new electric equilibrium in the three islands A, B and C as determined by $RS_2()$ is set to a very low electric power flowing through the lines.

The described scenario, although very simple, is a good example of how EPSyS allows to analyse the interactions between EI and II. In particular, our case study focused on the expected cumulated reward $Y_{[0,t]}$ and highlighted some critical timing interactions; those interactions are correlated to discontinuities (in this case, drops) in the plot of the evaluated measure. The obtained results are useful to understand the interplay between control operations and failures of the electric grid power lines from the timing point of view, and therefore take appropriate decisions to mitigate blackout events.

The analysis of the evolution of the electric power through the power lines is useful to better understand the interactions and their consequences, and to better reconstruct the chain of events. Of course, this kind of analysis can be applied to requested loads and generators as well, to observe the evolution of their values under the assumed failure events and II control operations.

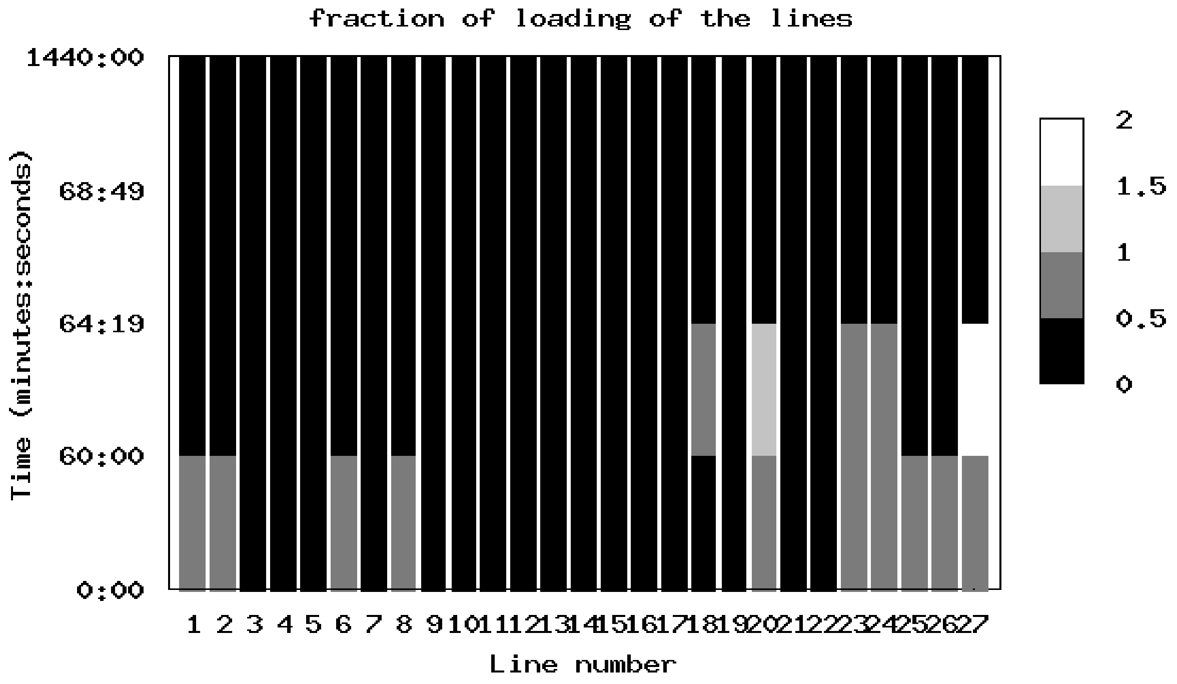


Figure 5-5: Evolution of the load through the lines for the $RS_2()$ delay=4:30 m

6 CONCLUSIONS

In this deliverable we have reported on four different preliminary validation and evaluation activities: definition of simple, but appropriate, models for the attacks, evaluation and correctness validation of the CIS (with and without recovery policy), and on the study of interdependences between II and EI.

We have presented experimental results based on data collected during a four year observation period from a large set of identically configured honeypots that have been deployed on the Internet. In particular, we proposed a methodology to process the collected data in order to identify statistical distributions that best characterize the times between attacks observed on the different platforms, taking into account the possible presence of silence periods that are due e.g., to the unavailability of the honeypots. The experimental results show that a mixture Pareto and Weibull distribution is well suited to describe the attack processes observed on several honeypot platforms. This result confirms the preliminary investigations derived in [Kaâniche *et al.* 2006] based on a small subset of the data presented in this section.

The obtained results should be useful to generate synthetic workloads that are representative of malicious traffic observed on the honeypots and to support the elaboration of quantitative security assessment models.

Concerning the evaluation of the CIS architecture, the following activities are planned during the last year of the project: i) refinement of the PRRW model presented in this deliverable, aiming to relax some assumptions (e.g. urgent reactive recoveries are actually delayed until the beginning of the next recovery slot) and to perform other sensitivity analyses (e.g. how the omission and intrusion rates impact on the system failure probability and availability); ii) definition of the model of some alternative recovery schema, in order to perform quantitative comparisons among the available schemas. Also the verification of correctness of the CIS architecture will continue to include more realistic hypothesis on the environment and the more recent evolution of CIS towards proactive-reactive recovery strategy (the CIS already considered in DEEM, but from a correctness point of view)

Concerning the quantitative evaluation of the impact of intedependencies, several activities are planned for the last year of the project. First, more extensive analyses are necessary, both on increasingly complex testbeds, like the IEEE Testbed 118, and on more realistic control scenarios taken from the project itself (the control system scenarios developed in the context of WP1). Moreover, new features are planned to be implemented in the simulator, mainly the repair process of permanently faulty components, which brings back the repaired component in full operation. Also, more sophisticated fault patterns are planned to be considered, affecting both EI components as well as II ones. Finally, the quantitative evaluation will be performed also by exploiting the modelling framework under development in WP2. Especially, the two evaluation approaches will be (at least in part) exercised on the same case studies, to perform cross-validation between the two methods, and for cross-fertilization.

REFERENCES

[Abou El Kalam *et al.* 2003] A. Abou El Kalam, E. E. Baida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Miege, C. Saurerel and G. Trouessin, “Organization based access control”, in *4th IEEE Int. Workshop on Policies for Distributed Systems and Networks (Policy03),* 2003.

[Abou El Kalam *et al.* 2007] A. Abou El Kalam, A. Baina, H. Beitollahi, A. Bessani, A. Bondavalli, M. Correia, A. Daidone, G. Deconinck, Y. Deswarte, F. Grandoni, N. Neves, T. Rigole, P. Sousa, P. Veríssimo, *Preliminary Architecture Specification.* Critical Utility InfrastructurAL Resilience. Project co-funded by the European Commission within the Sixth Framework Programme. Deliverable D4, 2007.

[Bondavalli *et al.* 2000] A. Bondavalli, I. Mura, S. Chiaradonna, R. Filippini, S. Poli and F. Sandrini, “DEEM: a tool for the dependability modeling and evaluation of multiple phased systems”, in *Int. Conference on Dependable Systems and Networks (DSN2000),* (New York, USA), pp.231-236, IEEE Computer Society, 2000.

[Donatelli *et al.* 2008a] S. Donatelli, E. Alata, J. Antunes, M. Kaaniche, V. Nicomette, N. F. Neves and P. Verissimo, *Experimental validation of architectural solutions.* Critical Utility InfrastructurAL Resilience. Project co-funded by the European Commission within the Sixth Framework Programme. Deliverable D26, 2008a.

[Donatelli *et al.* 2008b] S. Donatelli, S. Chiaradonna, D. Codetta Raiteri, F. Di Giandomenico, G. Franceschinis, M. Gribaudo, M. Kaâniche, P. Lollini, F. Romani and J. Sproston, *List of requirements on formalisms and selection of appropriate tools.* Critical Utility InfrastructurAL Resilience Project co-funded by the European Commission within the Sixth Framework Programme. Deliverable D11, 2008b.

[Garrone et al. 2007] F. Garrone, C. Brasca, D. Cerotti, D. Codetta Raiteri, A. Daidone, G. Deconinck, S. Donatelli, G. Dondossola, F. Grandoni, M. Kaâniche and T. Rigole, *Analysis of New Control Applications.* Critical Utility InfrastructurAL Resilience. Project co-funded by the European Commission within the Sixth Framework Programme. Deliverable D2, 2007.

[Hodge *et al.* 2004] V. Hodge and J. Austin, “A Survey of Outlier Detection Methodologies”, *Artifical Intelligence Review*, 22 (2), pp.85-126, 2004.

[Kaâniche *et al.* 2006] M. Kaâniche, E. Alata, V. Nicomette, Y. Deswarte and M. Dacier, “Empirical Analysis and Statistical Modeling of Attack Processes based on Honeypots”, in *WEEDS 2006 - workshop on empirical evaluation of dependability and security (in conjunction with the international conference on dependable systems and networks, (DSN2006),* pp.119-124, 2006.

[Kaâniche *et al.* 2008] M. Kaâniche, M. Beccuti, C. Brasca, S. Chiaradonna, S. Donatelli, F. Di Giandomenico, G. Franceschinis, K. Kanoun, J.-C. Laprie, P. Lollini and F. Romani, *Preliminary modelling framework. Critical Utility InfrastructurAL Resilience. Project co-funded by the European Commission within the Sixth Framework Programme. Deliverable D8*, 2008.

[Mura *et al.* 2001] I. Mura and A. Bondavalli, “Markov Regenerative Stochastic Petri Nets to Model and Evaluate the Dependability of Phased Missions”, *IEEE Transactions on Computers*, 50 (12), pp.1337-1351, 2001.

[Neves *et al.* 2008] N. F. Neves, P. Verissimo, A. Abou El Kalam, A. Baina, H. Beitollah, A. Bessani, A. Bondavalli, M. Correia, A. Daidone, G. Deconinck, Y. Deswarte, F. Garrone, F. Grandoni, H. Moniz, T. Rigole and P. Sousa, *Preliminary Specification of Services and Protocols.* Critical Utility InfrastructurAL Resilience. Project co-funded by the European Commission within the Sixth Framework Programme. Deliverable D10, 2008.

[Pouget *et al.* 2005] F. Pouget, M. Dacier and V.-H. Pham, *"Leurré.com: On the Advantages of Deploying a Large Scale Distributed Honeypot Platform",* E-Crime and Computer Conference (ECCE '05), Monaco, 2005.

[Provos *et al.* 2007] N. Provos and T. Holz, *Virtual Honeypots — From Botnet Tracking to Intrusion Detection,* Addison-Wesley, Boston, MA, USA, 2007.

[Sousa *et al.* 2007] P. Sousa, A. Bessani, M. Correia, N. F. Neves and P. Verissimo, "Resilient intrusion tolerance through proactive and reactive recovery", in *13th IEEE Pacific Rim Dependable Computing conference (PRDC-2007),* (Melbourne, Australia), 2007.

[Spitzner 2002] L. Spitzner, *Honeypots: Tracking Hackers,* Addison-Wesley, Boston, 2002.

[UML] UML, *"Unified Modelling Language (UML) web site, http://www.uml.org"*.

[Vanderviere *et al.* 2004] E. Vanderviere and M. Hubert, "An adjusted boxplot for skewed distributions", in *16th Symposium of IASC (COMPSTAT'04),* (Pragues, Czeck republic), 2004.