IMT Institute for Advanced Studies, Lucca

Lucca, Italy

On the Analysis and Evaluation of Trust and Reputation Systems

PhD Program in Computer Science and Engineering

XXV Cycle

By

Alessandro Celestini

2013

The dissertation of Alessandro Celestini is approved.

Program Coordinator: Prof. Rocco De Nicola, IMT Institute for Advanced Studies, Lucca

Supervisor: Prof. Rocco De Nicola, IMT Institute for Advanced Studies, Lucca

Co-supervisor: Prof. Michele Boreale, Università di Firenze, Italy

Co-supervisor: Francesco Tiezzi, IMT Institute for Advanced Studies, Lucca

Tutor: Francesco Tiezzi, IMT Institute for Advanced Studies, Lucca

The dissertation of Alessandro Celestini has been reviewed by:

,

,

IMT Institute for Advanced Studies, Lucca

2013

Abstract

In recent years, we have witnessed an increasing use of trust and reputation systems in different areas of ICT. The idea at the base of trust and reputation systems is of letting users to rate the provided services after each interaction. Other users may use aggregate ratings to compute reputation scores for a given party. The computed reputation scores are a collective measure of parties trustworthiness and are used to drive parties interactions.

Due to the widespread use of reputation systems, research work on them is intensifying and several models have been proposed. This calls for a methodology for the analysis and the evaluation of trust and reputation systems that can help researcher and developers in studying, designing and implementing such systems. In this thesis we propose different kinds of theoretical results and software tools that could be useful means for researchers and developers in area of trust and reputation systems.

Our work addresses the three main stages of trust and reputation systems development, namely study, design and implementation. We provide: 1) a general framework based on Bayesian decision theory for the assessment of trust and reputation models, 2) an analysis methodology for reputation systems based on a coordination language, 3) a software tool for network-aware evaluation of reputation systems and their rapid prototyping.

Contents

List of Figures

List of Tables

Acknowledgements

The preparation of this thesis is the last step of an experience began on March 2010 at IMT Institute for Advanced Studies, Lucca. In this period I had the chance to meet a lot of people and to have with them interesting discussions.

First of all I must thank Prof. Rocco De Nicola, my supervisor, for his constant support and the useful advices. The same shall be said about Prof. Michele Boreale and Francesco Tiezzi my co-supervisors, with whom I had the pleasure to work. Many other people shall be thanked, Prof. Hanne Riis Nielson and Prof. Flemming Nielson, who hosted me for few months in their research group at Technical University of Denmark. With them and their collaborator I had the opportunity to have useful and interesting discussions that have been fruitful for my research. Hernán Melgratti, who hosted me at the Computer Science Department of the Univeristy of Buenos Aires, for his useful suggestions.

A special thanks goes to my family, that has constantly supported me.

Declaration

Part of the material presented in this thesis has been previously published in some co-authored papers. In particular, Chapter 3 is based on [BC13], a joint work with Michele Boreale. Chapters 4 and 5 are instead based on [CDT13b] and [CDT13a], both co-authored by Rocco De Nicola and Francesco Tiezzi.

Vita

May 4, 1983	Born, Viterbo, Italy
May, 2006	Bachelor Degree in Computer Science La Sapienza, University of Roma, Italy
September, 2007 - February, 2008	Exchange Student, Erasmus Programme KTH Royal Institute of Technology, Stockholm, Sweden
December, 2008	Bachelor Degree in Computer Science La Sapienza, University of Roma, Italy
January, 2009 - March, 2010	Software System Designer CASPUR, Roma, Italy
March 2010	PhD Student, IMT Lucca, Italy.
March, 2012 - July 2012	Visiting Research Period DTU Technical University of Denmark Kongens Lyngby, Denmark
November, 2013 - December, 2013	Visiting Research Period University of Buenos Aires, Buenos Aires, Argentina

Publications

1. M. Boreale and A. Celestini, "Asymptotic Risk Analysis for Trust and Reputation Systems", in *SOFSEM 2013: Theory and Practice of Computer Science*, Lecture Notes in Computer Science Volume 7741, 2013, pp 169-181, Springer Berlin Heidelberg.
2. A. Celestini, R. De Nicola, and F. Tiezzi, "Specifying and Analysing Reputation Systems with a Coordination Language", in *Proceedings of 28th Annual ACM Symposium on Applied Computing (SAC)*, ACM press, 2013, pp 1363-1368.
3. A. Celestini, R. De Nicola, and F. Tiezzi, "Network-aware Evaluation Environment for Reputation Systems", in *Trust Management VII – 7th IFIP WG 11.11 International Conference*, IFIPTM, 2013.

Presentations

Conference talks:

1. "Specifying and Analysing Reputation Systems with a Coordination Language", *28th Symposium On Applied Computing*, Coimbra, Portugal, March, 2013.
2. "Asymptotic Risk Analysis for Trust and Reputation Systems", *39th International Conference on Current Trends in Theory and Practice of Computer Science*, Špindlerův Mlýn, Czech Republic, January, 2013.

Other talks:

3. "Analysis for Trust and Reputation Systems", *CINA Kick-off Meeting*, Pisa, Italy, February, 2013.
4. "Asymptotic Risk Analysis for Trust and Reputation Systems", University of Buenos Aires, Buenos Aires, Argentina, December, 2012.
5. "A Framework for the Analysis of Trust and Reputation Systems", *16th MT-LAB Workshop*, DTU Technical University of Denmark, Kongens Lyngby, Denmark, April, 2012.
6. "A Framework for Network-aware Evaluations of Reputation Systems", DTU Technical University of Denmark, Kongens Lyngby, Denmark, March, 2012.

Chapter 1

Introduction

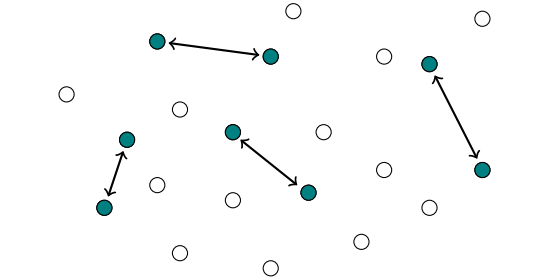
The concept of trust is an integrative component of human society. In our daily activities we establish trust-based interactions with others in several contexts. As an example of these interactions we take here the sale of goods. Both in the case of real or virtual markets, in sales we have two parties, the seller and the buyer, each one with different goals. The seller has to trust the buyer, which means that the buyer will pay, that the payment method will be valid, etc. The buyer has to trust the seller, which means that the seller will send the goods by the time stated, that the quality of the goods will be the one stated, etc. Living in a social world, becoming smaller due to networking technologies, interactions are unavoidable, in everyday lives we are constantly interacting with something or someone. In such interactions we always deal with the issue of how to evaluate others' trustworthiness.

In this thesis our aim is more specific, we are interested in trust interactions taking place among parties in distributed systems. Such parties can be human beings using devices or just devices communicating and executing code. In computer science, there are many definitions and models for trust management (see, e.g., those reported in [BFIK99; ZM00; JIB07; SS05]). Such issue is usually tackled in the field of computer security and, on the basis of the mechanism used for trust management, we talk of *hard* or *soft* security mechanisms [RJ96].

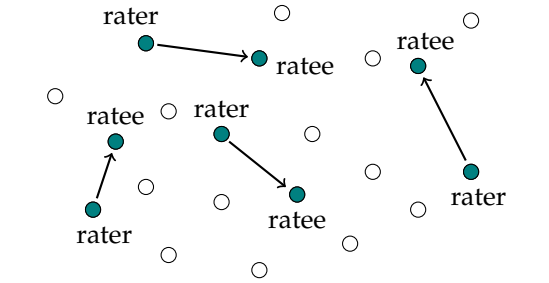
Trust and reputation systems are a specific approach to trust management. Such systems are used as decision support tools for different applications in several contexts. In recent years, we have seen an increasing use of them in different areas of ICT, from e-commerce to different forms of open computer networking. This phenomenon is likely to continue, due to the success of networked applications (like social networks or other Web 2.0 technologies) and to the need, in such environments, of instruments to build up relationships of trust among interacting parties.

Probably, the best known applications making use of trust and reputation systems are those related to e-commerce: well-known examples in this context are the auction site eBay, the online shop Amazon, and software application stores, like Google Play and Apple App Store. However, trust management systems are used in many other contexts and applications, where huge amount of data related to reputations of peers are usually available, such as ad-hoc networks [NCL07], P2P networks [XL04; WV03] and sensor networks [BXEK07]. Parties, which are willing to interact in these environments, are likely to be disconnected from their preferred security infrastructures and/or have no trusted information about their partners. Thus, they have to rely on other techniques to build up relationships of trust among each other.

The idea at the base of trust and reputation systems is to let their users, the *raters*, rate the services providers, the *ratees*, after each interaction. For instance, rating values could refer to the quality of a service, or to the success of the interaction. Figure 1 graphically depicts such generic scenario where a trust and reputation system is in use: parties active in the system freely interact (Figure 1 (a)) and rate each other after each interaction (Figure 1 (b)). Then, other users or the parties themselves may use aggregate ratings to compute reputation scores for a given party. The computed *reputation* scores are a collective measure of parties' trustworthiness and are used to drive parties' interactions, i.e. a party selects the party to interact with on the basis of its reputation score. This approach to trust management is referred to as *computational trust*. In computational trust parties' trustworthiness is evaluated on the basis of parties' past behaviour, whereas *credential-based* approaches [EFL+99; NT94] rely



(a) Interactions between parties



(b) Rating release

Figure 1: A generic scenario in which a reputation system is in use

on access control policies and/or use of certificates for evaluating parties' trustworthiness.

In the following we briefly present the two categories of trust management approaches, namely credential-based and computational trust.

1.1 Credential-based Trust

Credential-based approaches are based mainly on two concepts: *authentication* and *authorization*. In such approaches the aim is to authenticate parties and then check if parties are allowed to perform the actions they are trying to carry out. Indeed, we can say that parties' trustworthiness

is evaluated on the basis of parties' credentials. The assumptions are that all parties interacting in the systems can be authenticated and that, for each of them, there exists a policy stating which actions it is authorized to perform. Two issues immediately arise, first all parties have to be known in advance in order to be authenticated. Second, a policy rule for each of them should be set.

In distributed systems the first issue, about parties authentication, is usually addressed through public key cryptography. In public key cryptography each party in the system is assumed to own a key pair <*public key, private key*>. The private key is known only by the owner of the pair, while the public key is known by everyone. Below, we present the general idea at the base of public key authentication protocols. If party A wants to authenticate party B, assuming A knows B's public key, the basic protocol run by the two parties works as follows: let $< e_B, d_B >$ the key pair of B and r a random number chosen by A. Party A sends r encrypted with B's public key e_B to B. Party B proves it knows d_B, that is its identity, by decrypting the message and sending back r to A. In a system with thousand of parties the question is how to securely manage thousand of public keys. Such issue is usually addressed trough the use of a public key infrastructure (PKI) [KPS02; TW10]. The basic components of a PKI are: certificates, a repository for retrieving certificates, a method of revoking certificates, and a method of evaluating a chain of certificates.

For the second issue, concerning policy rules stating which actions parties can perform, there exist several access control models enabling to control access to data, resources and systems [NIST09]. The oldest and most basic form of access control are the Access Control Lists (ACLs). An ACL states for each resource in the system which are the parties that can access it and which actions they can perform on it. In addition to ACLs there are several others models, such as Role-based Access Control (RBAC), Attribute Based Access Control (ABAC) and Policy-Based Access Control (PBAC). All these models try to address shortfalls of the others and are tailored for specific environments, thus the choice of the access control model should be made on the basis of systems' features.

Below, as reference standards in traditional trust, we briefly remind Kerberos, X.509, PGP, SSL/TLS and XACML.

Kerberos. Kerberos is a secret key based service for providing authentication in a network, where the access to remote resources is granted to users after authentication. In Kerberos, user's workstation performs the authentication protocol on user's behalf and the network itself is assumed to be insecure. Two servers are used to manage authentication and resource access, the Authentication Sever and the Ticket-Granting Server. We refer the reader to [KPS02; TW10; NT94] for an extensive discussion of Kerberos.

X.509 and PGP. The X.509 and PGP standards are two well-known certificate systems, they mainly differ for the way public keys are certified and how certificate chains are verified. The standard X.509 assumes existence of entities called certification authorities, which release certificates and sign them. Instead, with PGP, each user generates its pair of keys and decides which keys to trust. New trusted keys are got directly from new users or introduced by trusted users. We refer the reader to [KPS02; TW10] for an extensive discussion of X.509 and PGP.

SSL/TLS. The SSL/TLS standard is used to establish secure network connections. It is broadly used in web applications, mainly for e-commerce transactions. Indeed, the use of HTTP over SSL, called HTTPS, is the most common application of the SSL/TLS protocol. We refer the reader to [KPS02; TW10] for an extensive discussion of the SSL/TLS protocol.

XACML. The eXtensible Access Control Markup Language (XACML) is a general-purpose access control policy language implemented in XML. XACML was developed to specify access control policy in a machine-readable format. With XACML it is possible to implement ABAC systems or RBAC systems as a specialization of ABAC systems. XACML is also used in PBAC systems, in such systems policies creation

can be complicated and XACML is a useful means for creating, specifying, and enforcing effective access control policies. We refer the reader to [XACML; NIST09] for an extensive discussion of XACML and access control models.

However, credential-based approaches are not well suited for open environments, where the sets of parties and resources dynamically change in time.

1.2 Computational Trust

In computational trust approaches, parties' trustworthiness is evaluated on the basis of the parties' past behaviour. The assumption at the base of such approaches is to have a distributed system where several parties freely interact. After each interaction, *raters* rate the services providers, then the rater itself, or other parties, may use such ratings to compute reputation scores for a given party. Party's reputation is the synthetic parameter estimating the party's behaviour and is used to evaluate parties trustworthiness.

Depending on the computational trust model in use, the ratings released by parties can assume values in different domains. There are models in which each interaction can be evaluated using just two values, representing the success and the failure of the interaction [JI02; DA04]. Another possibility is using rating values in an interval of n values, with each rating representing the quality of the service provided [JH07; DA04]. There are also models where instead of numerical values, attributes are used to rate parties, e.g. in [ARH00] the following attributes are used: very trustworthy, trustworthy, untrustworthy, very untrustworthy.

Trust and reputation are often used as synonyms in the literature. In our work we comply with the distinction made in [JIB07]. According to [JIB07], trust is a subjective perception of reliability of a party, mainly derived from private knowledge and/or belief (e.g., direct interactions with the party). Instead, reputation is an objective measure of partys

trustworthiness derived from referrals or ratings provided by other parties. Parties can trust other parties despite their reputation values, this is due to the subjectivity property of trust. Indeed reputation systems are used as decision support tools where reputation values are used to drive parties interactions.

More specifically, in our work we focus on probabilistic trust [SKN07; KNS08; MMH02; TPJL06; JI02], which represents a specific approach to computational trust. The basic postulate of probabilistic trust is that the behaviour of each party can be modeled as a probability distribution, drawn from a given family, over a certain set of interaction outcomes (success/failure being the simplest case). In this approach the task of computing reputation scores reduces to inferring the true distribution's parameters for a given party. The information about party's past behaviour is used for such inference, i.e. rating values are treated as statistical data in the inference process.

1.3 Contribution

Due to the widespread use of reputation systems, research work on them is intensifying and several models have been proposed [JIB07; LLYY09; SS05; MGM06]. Thus, once a reputation system has to be designed several choices have to be made at different levels of the development process, before its deployment in a network environment. This calls for a methodology for the analysis and the evaluation of trust and reputation systems that can help researchers and developers in studying, designing and implementing such systems. We address this challenge by proposing different kinds of theoretical and software frameworks and tools that, in our opinion, can support the development of trust and reputation systems.

The main contributions of our work can be summarized as follows:

1. **Theoretical Framework:** a general framework based on Bayesian decision theory for the theoretical assessment of trust and reputation models.

2. **Analysis Methodology:** a methodology for analysing reputation systems based on a coordination language.

3. **Software Tool:** a software environment for network-aware evaluation of reputation systems and their rapid prototyping.

Theoretical Framework. Whenever existing or new reputation models have to be analysed, a theoretical framework for the assessment of such models is needed. In this phase it is interesting to study the models on the basis of a small set of simple parameters, such as quantity of available information and decision strategies, while abstracting from implementation and deployment details. To this aim, we propose a general framework based on Bayesian decision theory for the assessment of trust and reputation systems. Within our theoretical framework we study how to quantify the *confidence* in the decisions calculated by the system. We analyse how this confidence is related to parameters as decision strategy and number of available ratings. We analyse if there are optimal strategies that maximize confidence when additional information becomes available.

In our analysis, we study the behaviour of trust and reputation systems by relying on the concept of *loss* function; a loss function evaluates the consequences of possible decisions taken by the system associating a loss to each decision. We quantify the *confidence* in the decisions calculated by trust and reputation systems in terms of *risk* quantities based on *expected (also known as bayes)* and *worst-case* loss. We study the behaviour of these quantities with respect to the available information, that is the number of available rating values and the decision strategy, in the case of independent and identically distributed observations. We show that there are optimal strategies that maximize confidence as more and more information becomes available. Finally, we study an extention of our framework to a class of rating mechanisms where each rater is characterised by a (unobservable, possibly malicious) bias. This can lead the rater to under- or over-evaluate its interactions with the ratees.

Analysis Methodology. Concerning the integration of reputation systems with end-user applications, a methodology for tuning trust and reputation models in order to fit to the characteristics of the given network environment is needed. In particular, it is interesting to study whether in the phase of models tuning, the features of the original models are kept. We address such issues by proposing a verification methodology based on the use of the coordination language KLAIM [BBD+03; DFP98] and related analysis tools [DKL+07; Lor10]. Such approach enables verification of reputation system specifications. Specifically, it is possible to check whether trust and reputation models meet the expected behaviour, how parties' initials reputations affect the models and how parties' behaviours affect their reputations. In our study, we first define a parametric KLAIM specification of a reputation system that can be instantiated with different reputation models. Then, we consider a stochastic specification obtained by considering actions with random (exponentially distributed) duration. The resulting specification allows us to perform quantitative analysis of estimation properties of the considered system.

Software Tool. The last issue we address is related to implementation, i.e. to the phase when reputation systems have to be deployed and tested in real network environments. At this stage, real-word implementation details of trust and reputation systems and of the network environment where they have to be deployed have to be taken into account in the evaluation. We have developed a software tool (NEVER) for network-aware evaluation of reputation systems and for their rapid prototyping through experiments performed according to user-specified parameters. On the one hand, NEVER provides a framework for rapidly developing Java-based implementations of reputation system models and for easily configuring different networked execution environments on top of which the reputation systems will run. On the other hand, NEVER can be used for automatically performing experiments on the reputation system implementations according to user-specified parameters; this enables the study of their behaviour while executing on given network infrastructures.

Overall our contribution addresses issues related to the study, the design and the implementation of trust and reputation systems. Indeed, we provide theoretical and software tools for the analysis and evaluation of trust and reputation systems, at different stages of their development.

1.4 Structure of the thesis

The rest of the thesis is organized as follows. In Chapter 2 we briefly introduce the technical concepts used throughout the thesis. We first recall some basic notions of Probability Theory, Information Theory and discuss some relationships between Information Theory and Statistics. We describe the basic concepts of Bayesian Decision Theory and, introduce the coordination language KLAIM and related analysis tools. Then, we give an overview of probabilistic trust approaches by describing some of the models proposed in the literature. Finally, we show some cases of reputation systems that are successfully used in real applications.

In Chapter 3 we present our general framework, based on Bayesian decision theory, for the assessment of trust and reputation models. We introduce the framework and discuss our results on the analysis of such systems. We close the chapter by presenting an extention of the framework for different data models, with rating values given in different ways.

In Chapter 4 we introduce a verification approach for reputation systems that is based on the use of the coordination language KLAIM and related analysis tools. We define a parametric KLAIM specification of a reputation system that can be instantiated with different reputation models. Then, we consider stochastic specifications enabling quantitative analysis of properties of the considered system. Finally, we present verification results on some reputation systems.

In Chapter 5 we present NEVER, a software tool for network-aware evaluation of reputation systems and their rapid prototyping. The NEVER evaluation of reputation systems is carried out through experiments performed according to user-specified parameters. In such experiments the networked execution environment is explicitly taken into

account. We close the chapter by presenting some evaluation results obtained with our tool.

In Chapter 6 we comment on the research results presented in the thesis by also comparing them with more closely related work. We summarize the main contributions and propose possible directions for further research.

Chapter 2

Preliminaries

In this chapter we briefly introduce the technical concepts used in the rest of the thesis. In Section 2.1 we recall some basic notions of Probability Theory. In Section 2.2 we introduce Information Theory and Statistics and show their relationship. In Section 2.3 we describe the basic concepts of Bayesian Decision Theory and in Section 2.4 we introduce the coordination language KLAIM and related analysis tools. Finally, in Section 2.5 we give an overview of probabilistic trust approaches and in Section 2.6 we describe the operation of some reputation systems used in real applications.

2.1 Probability Theory

Probability theory is technically a branch of measure theory, it reasons about chance, uncertainty, likelihood of phenomena. In this section, we first introduce few notions of measure theory then we recall some basic notions of probability theory. We refer the reader to [Wil91; Sti99; GS01; Kal02] for an extensive presentation of these topics.

Below we provide the definitions of σ-$field$, *measure space*, *measurable set* and *measure* to then introduce some basic notions of probability theory.

Definition 2.1.1 *Let Ω be a set, a σ-$field$ (or σ-$algebra$) on Ω is a collection*

$\mathcal{F}$ of subsets of Ω satisfying the following conditions:

(a) $\emptyset \in \mathcal{F}$,

(b) $A \in \mathcal{F} \Rightarrow A^c \in \mathcal{F}$, *where* A^c *denotes the complement set,*

(c) $A_1, A_2, ..., A_n \in \mathcal{F}$ *with* $n \in \mathbb{N} \Rightarrow \bigcup_n A_n \in \mathcal{F}$

The smallest σ-$field$ associated with Ω is the collection $\mathcal{F} = \{\emptyset, \Omega\}$ and the largest one is the *power set* of Ω, written 2^{Ω}. A *measurable set* is a pair $(\Omega, \mathcal{F})$, where Ω is a set and $\mathcal{F}$ is a σ-$field$ on Ω.

Definition 2.1.2 *Let* $(\Omega, \mathcal{F})$ *a measurable set, a* measure *on* $(\Omega, \mathcal{F})$ *is a function* $\mu : \mathcal{F} \to [0, \infty)$ *such that:*

(a) $\mu(\emptyset) = 0$,

(b) $\mu(A) = \sum_n \mu(A_n)$, *where* $A_n (n \in \mathbb{N})$ *is a sequence of pairwise disjoint sets in* $\mathcal{F}$ *with union* $A = \bigcup_n A_n$.

The triple $(\Omega, \mathcal{F}, \mu)$ is then called a *measure space*.

A *probability measure* $\mathbb{P}$ on $(\Omega, \mathcal{F})$ is a measure such that $\mathbb{P}(\Omega) = 1$. The triple $(\Omega, \mathcal{F}, \mathbb{P})$ is called *probability space* and the set Ω is called *sample space*. A point ω of Ω is called a *sample point* or *outcome*. The collection $\mathcal{F}$ is called *family of events* and an *event* is an element of $\mathcal{F}$. If A and B are two events and $\mathbb{P}(B) > 0$, then the *conditional probability* that A occurs given that B occurs is defined as

$$\mathbb{P}(A|B) = \frac{\mathbb{P}(A \cap B)}{\mathbb{P}(B)} \tag{2.1}$$

We denote with $\mathbb{P}(A|B)$ the conditional probability and we read "the probability of A given (or conditioned on) B". It is not always the case that the occurrence of an event B changes the probability that another event A occurs. If the conditional probability $\mathbb{P}(A|B)$ remains unchanged, i.e. $\mathbb{P}(A|B) = \mathbb{P}(A)$, then we say that A and B are *independent*. More formally, events A and B are called independent if

$$\mathbb{P}(A \cap B) = \mathbb{P}(A)\mathbb{P}(B). \tag{2.2}$$

Experimental outcomes are not always numerical, but it is often better to work with numbers than with outcomes in the original sample space. We can assign a number to any outcome $\omega \in \Omega$ using *random variables*. A random variable is a real-valued function $X : \Omega \to \mathbb{R}$ such that $\{\omega \in \Omega : X(\omega) \leq x\} \in \mathcal{F}$ for each $x \in \mathbb{R}$. We can think of a random variable just as a function mapping Ω in $\mathbb{R}$. We denote random variables with upper-case letters, such as X, Y, Z and their possible numerical values with lower-case letters, such as x, y, z. A distribution function is associated with every random variable. A *distribution function* of a random variable X is a function $F : \mathbb{R} \to [0, 1]$ such that $F(x) = \mathbb{P}(X \leq x)$, where $\{X \leq x\}$ denotes the event $\{\omega \in \Omega : X(\omega) \leq x\}$. We denote with F_X the distribution function of the random variable X.

A random variable X is called *discrete* if it takes values only in some countable subset $\mathcal{X}$ of $\mathbb{R}$. We denote with calligraphic letters $\mathcal{X}, \mathcal{Y}, \mathcal{Z}$ possible subsets of $\mathbb{R}$. The *probability mass function* of X is $p : \mathbb{R} \to [0, 1]$ such that $p(x) = \mathbb{P}(X = x)$. A random variable X is called *continuous* if it takes values in some uncountable subset $\mathcal{X}$ of $\mathbb{R}$ and if its distribution function can be expressed as $F(x) = \int_{-\infty}^{x} p(u)du$, for some integrable function $p(x)$ called the *probability density function* of X.

Let X be a discrete random variable, its expected value, denoted by $\mathbb{E}[X]$, is defined as

$$\mathbb{E}[X] = \sum_{x} x\mathbb{P}(X = x) \tag{2.3}$$

Notation: Let X be a random variable taking values in $\mathcal{X}$, we say that X is distributed according to a probability distribution $p(\cdot)$ if for each $x \in \mathcal{X}$, $\mathbb{P}(X = x) = p(x)$, and we write $X \sim p(\cdot)$. We will use the term probability distribution both denoting probability mass function and probability density function, the use will be clear by the context. The *support* of $p(\cdot)$ is defined as $\text{supp}(p) = \{x \in \mathcal{X} : p(x) > 0\}$. We let $p^n(\cdot)$ denote the n-th extension of $p(\cdot)$, defined as $p^n(x^n) = \prod_{i=1}^{n} p(x_i)$, where $x^n = (x_1, x_2, ..., x_n)$; this is in turn a probability distribution on the set $\mathcal{X}^n$. For any $A \subseteq \mathcal{X}$ we let $p(A)$ denote $\sum_{x \in A} p(x)$. When $A \subseteq \mathcal{X}^n$ and n is clear from the context, we shall abbreviate $p^n(A)$ as just $p(A)$.

2.2 Information Theory and Statistics

Information theory reasons about quantification of information. The fundamental measure of information is called *entropy*, which quantifies the self-information of a random variable, i.e. the uncertainty involved in predicting its value. We refer the reader to [CT06] for an extensive presentation of the topic.

Let X be a discrete random variable taking value in set $\mathcal{X}$ and probability mass function $p(x) = \mathbb{P}(X = x), x \in \mathcal{X}$. The *entropy* $H(X)$ of X is defined as

$$H(X) = -\sum_{x \in \mathcal{X}} p(x) \log p(x) \tag{2.4}$$

The $\log$ is to the base 2 and entropy is expressed in bits. The entropy of a random variable measures the average amount of information required to describe the random variable.

Given two distributions p and q on the same set $\mathcal{X}$, the relative entropy is a measure of the distance between these two distributions. The *relative entropy*, or *Kullback-Leibler distance* $D(p||q)$, between two probability mass functions $p(x)$ and $q(x)$ is defined as

$$D(p \parallel q) = \sum_{x \in \mathcal{X}} p(x) \log \frac{p(x)}{q(x)} \tag{2.5}$$

with the convention that $0 \log \frac{0}{0} = 0$, $0 \log \frac{0}{q} = 0$ and $p \log \frac{p}{0} = \infty$. It can be shown that the Kullback-Leibler distance (KL-dis) is always nonnegative and is 0 if and only if $p = q$. $D(p \parallel q)$ it is not a true distance since it is not symmetric and does not satisfy the triangle inequality. The KL-dis $D(p \parallel q)$ is a measure of the inefficiency of assuming that a distribution is q when the true distribution it is actually p.

We now introduce some concepts at the basis of the relationship between information theory and statistics. We then consider the problem of hypothesis-testing and which is the best possible error exponents for such tests. Let $x^n = x_1, ..., x_n$ a sequence of n elements from a set $\mathcal{X}$, with $n > 0$, and $a \in \mathcal{X}$. We denote with $N(a, x^n)$ the number of occurrences of a in x^n. The *type* or *empirical probability distribution* of x^n is denoted by

t_{x^n}, and is the relative proportion of occurrences of each element

$$t_{x^n}(a) = \frac{N(a, x^n)}{n} \text{ for all } a \in \mathcal{X}. \tag{2.6}$$

The *type* t_{x^n} is a probability distribution on $\mathcal{X}$. Let $\mathcal{P}_n$ denote the set of types with denominator n, it is possible to show that the number of types is at most polynomial in n. Since the number of sequences is exponential in n, it follows that at least one type has exponentially many sequences in its type class.

Let Θ be a set of parameters: we let $\{p(\cdot|\theta)\}_{\theta\in\Theta}$ denote a parametrized family of probability distributions. When convenient, we shall denote a member of this family, $p(\cdot|\theta)$, as just p_θ. Given a sequence x^n that is a realization of n independent and identically distributed (i.i.d.) random variables $X^n = X_1, ..., X_n$, with $X_i \sim p(\cdot|\theta)$, a standard problem is to decide which of the distributions $p(\cdot|\theta)$ generated the data. This is a general hypothesis-testing problem, where the distributions $p(\cdot|\theta)$, $\theta \in \Theta$, are the hypotheses: the classical, binary formulation is given for $|\Theta| = 2$. We represent the decision-making process by a guessing function $g : \mathcal{X}^n \to \Theta$ and we define the error probability for an hypothesis θ as follows. For $n \geq 1$ and each θ, let $A_\theta^{(n)} = g^{-1}(\theta) \subseteq \mathcal{X}^n$ be the *acceptance region* for hypothesis θ (relatively to g) and let $A_\theta^{(n)^c}$ the complement set of $A_\theta^{(n)}$. Then the probability of error for θ is

$$P_\theta^{(g)}(n) = p(A_\theta^{(n)^c}) \,. \tag{2.7}$$

In a Bayesian framework, an a priori probability $\pi(\theta)$ is assigned to each hypothesis, and the overall error probability is defined as the average, assuming Θ is discrete:

$$P_e^{(g)}(n) = \sum_\theta \pi(\theta) P_\theta^{(g)}(n) \,. \tag{2.8}$$

It is well-known (see, e.g., [CT06]) that optimal strategies, i.e. strategies g minimizing the error probability $P_e^{(g)}(n)$, are obtained when g satisfies the *Maximum A Posteriori* (MAP) criterion (see Section 3.1.4 for a

formal definition of this criterion). In this case, provided $p(\cdot|\theta) \neq p(\cdot|\theta')$ for $\theta \neq \theta'$, it holds that as $n \to +\infty$, $P_e^{(g)}(n) \to 0$. What is also important, though, is to characterize *how fast* the probability of error approaches 0. Intuitively, we want to be able to determine an exponent $\rho \geq 0$ such that, for large n, $P_e^{(g)}(n) \approx 2^{-n\rho}$.

To this purpose, we introduce the notion of rate for a generic non-negative real-valued function f.

Definition 2.2.1 (Rate) *Let $f : \mathbb{N} \to \mathbb{R}^+$ be a nonnegative function. Assume $\gamma = \lim_{n\to\infty} f(n)$ exists. Then, provided the following limit exists, we define the following nonnegative quantity:*

$$\mathrm{rate}(f) = \lim_{n\to\infty} -\frac{1}{n} \log |f(n) - \gamma|.$$

This is also written as $|f(n) - \gamma| \doteq 2^{-n\rho}$, where $\rho = \mathrm{rate}(f)$.

Intuitively, $\mathrm{rate}(f) = \rho$ means that, for large n, $|f(n) - \gamma| \approx 2^{-n\rho}$. Note that we do allow $\mathrm{rate}(f) = +\infty$, a case that arises for example when $f(n)$ is a constant function.

The rate of decrease of $P_e^{(g)}(n)$ is given by *Chernoff Information*. Given two probability distributions p, q on $\mathcal{X}$, we let their Chernoff Information be

$$C(p, q) = -\min_{0\leq\lambda\leq 1} \log\Big(\sum_{x\in\mathrm{supp}(p)\cap\mathrm{supp}(q)} p^{\lambda}(x) q^{1-\lambda}(x)\Big) \tag{2.9}$$

where we stipulate that $C(p, q) = +\infty$ if $\mathrm{supp}(p) \cap \mathrm{supp}(q) = \emptyset$. Here $C(p, q)$ can be thought of as a sort of distance between p and q: the more p and q are far apart, the less observations are needed to discriminate between them. Assume we are in the binary case, $\Theta = \{\theta_1, \theta_2\}$ and let $p_i = p(\cdot|\theta_i)$ for $i = 1, 2$. Then a well-known result gives us the rate of convergence for the probability of error, with the proviso that $\pi(\theta_1) > 0$ and $\pi(\theta_2) > 0$ (cf. [CT06]): $P_e^{(g)}(n) \doteq 2^{-nC(p_1,p_2)}$ (here we stipulate $2^{-\infty} = 0$). Note that this rate does not depend on the prior distribution $\pi(\cdot)$ on $\{\theta_1, \theta_2\}$, but only on the probability distributions p_1 and p_2. This result extends to the case $|\Theta| < +\infty$, it is enough to replace $C(p_1, p_2)$ by $\min_{\theta\neq\theta'} C(p(\cdot|\theta), p(\cdot|\theta'))$, thus (see [LJ97; BPP11]):

$$P_e^{(g)}(n) \doteq 2^{-n \min_{\theta\neq\theta'} C(p(\cdot|\theta),p(\cdot|\theta'))} \tag{2.10}$$

with the understanding that, in the $\min$, $\pi(\theta) > 0$ and $\pi(\theta') > 0$.

2.3 Bayesian Decision Theory

Statistical decision theory concerns with the process of making decisions in presence of statistical knowledge. The general assumption is that the uncertainties involved in the decision process can be expressed as unknown numerical values, θ. Such values are called *world states* or *parameters*; we denote with Θ the set of all possible parameters. The statistical knowledge is used to obtain information about the parameters. In a Bayesian setting an other type of information is particularly relevant, the *prior information*. Such information comes from other source than the statistical investigation.

The main components of bayesian decision theory are three:

1. an a prior distribution $\pi(\cdot)$ over the world states Θ. Such distribution $\pi(\cdot)$ represents a prior information about the world states, in addition to sample information.

2. an observational model $p(\cdot|\cdot)$, that represents how data are generated in the world. The value $p(o|\theta)$ denotes the probability of observing $o \in \mathcal{O}$ when the world state is $\theta \in \Theta$.

3. a loss function $\mathrm{L}(\cdot, \cdot) : \Theta \times \mathcal{D} \to \mathbb{R}^+$, where $\mathcal{D}$ denote the decision set. The loss function $\mathrm{L}(\theta, d)$ gives the loss incurred when a decision $d \in \mathcal{D}$ is made and the state of the world turns out to be θ.

Determined the components of the model, the decision making process consists of making a decision d on the basis of a sequence of observations $o^n = o_1, ..., o_n$, with $n > 0$ and $o^n \in \mathcal{O}^n$. In bayesian decision theory the decision making process is formalized via decision functions. For any n, a decision function is a function $g : \mathcal{O}^n \to \mathcal{D}$.

The choice of an optimal decision relay on the concept of loss, that is computed through loss functions. Several standard types of loss can be considered, we report below some example of them. In Chapter 3 we use a Bayesian decision theory framework for our analysis of trust and

reputation systems. For an extensive presentation of Bayesian decision theory we refer the reader to [Ber85; Rob07].

2.3.1 Loss Functions

Loss functions evaluate the consequences of possible decisions, by associating a loss to each of them. Often it is not straightforward how this evaluation should be done, it depends on the system features. Below we list a few standard definitions of loss functions.

Linear Loss The loss

$$\mathrm{L}(\theta, d) = \begin{cases} k_0(\theta - d) & \text{if } (\theta - d) \geq 0, \\ k_1(d - \theta) & \text{if } (\theta - d) < 0 \end{cases}$$

is called *linear loss*. The constants k_0 and k_1 are usually different, they reflect the importance of underestimation and overestimation, respectively. When the two constants are equal the loss is equivalent to $\mathrm{L}(\theta, d) = |\theta - d|$, which is called *absolute loss*.

0-1 Loss The loss

$$\mathrm{L}(\theta, d_i) = \begin{cases} 0 & \text{if } \theta \in \Theta_i, \\ 1 & \text{if } \theta \in \Theta_j \ \ (j \neq i) \end{cases}$$

is called *0-1 loss*. In this case two decisions are possible, i.e. $\mathcal{D} = \{d_0, d_1\}$, and the loss associated to each decision can be only 0 or 1. It is 0 if a correct decision is made and 1 otherwise, e.g. d_0 is correct if $\theta \in \Theta_0$ and d_1 is correct if $\theta \in \Theta_1$. This loss is mainly used in the case of two-actions decision problems, e.g. hypothesis testing.

Squared-Error Loss The loss function $\mathrm{L}(\theta, d) = (\theta - d)^2$ is called *squared-error loss*. This loss function penalizes large errors. Anyway, this penalization could be considered too severe in some systems.

2.4 The KLAIM Coordination Language and Related Tools

In this section we informally present KLAIM [BBD+03; DFP98], a coordination language specifically designed for modelling mobile and distributed applications and their interactions, which run in a network environment. Then, we introduce the KLAIM's stochastic extension STOKLAIM [DKL+06a; DLM05; DKL+07], the stochastic logic MOSL [DKL+07; DKL+06b] and the analysis tool SAM [DKL+07; Lor10]. Finally, we introduce KLAVA [BDP02], a Java library implementing the run-time support for KLAIM actions.

2.4.1 KLAIM

In our presentation of the KLAIM language we consider a version of KLAIM enriched with standard control flow constructs (i.e., assignment, if-then-else, sequence, etc.). Such constructs were not included in the original presentation of the language [DFP98], however they can be easily rendered in KLAIM (by resorting, e.g., to choice, fresh names and recursion in the usual way) and are directly supported by related tools.

The syntax we use is reported in Table 1, where s, s',... range over *locality names* (i.e., network addresses); **self**, l, l',... range over *locality variables* (i.e., aliases for addresses); ℓ, ℓ',... range over locality names and variables; x, y,... range over *value variables*; X, Y,... range over *process variables*; e, e',... range over *expressions*[1]; A, B,... range over *process identifiers*[2]. We assume that the set of variables (i.e., locality, value and process variables), the set of values (locality names and basic values) and the set of process identifiers are countable and pairwise disjoint.

KLAIM *nets* are finite plain collections of nodes where *components*, i.e.

[1]The precise syntax of expressions is not specified here. Suffice it to say that expressions contain basic values (booleans, integers, strings, floats, etc.) and variables, and are formed by using the standard operators on basic values, simple data structures (i.e., arrays and lists) and the non-blocking retrieval actions **inp** and **readp** (explained in the sequel).

[2]We assume that each process identifier A with arity n has a unique definition, visible from any locality of a net, of the form $A(f_1, \ldots, f_n) \triangleq P$, where f_i are pairwise distinct. Notably, p_i and f_j denote actual and formal parameters, respectively.

(*Nets*)	N	::=	$\mathbf{0} \mid s ::_\rho C \mid N_1 \parallel N_2 \mid (\nu s)N$
(*Components*)	C	::=	$\langle t \rangle \mid P \mid C_1 \mid C_2$
(*Processes*)	P	::=	$\mathbf{nil} \mid a \mid P_1 ; P_2 \mid P_1 \mid P_2$
			$\mid \mathbf{if}\ (e)\ \mathbf{then}\ \{P\}\ \mathbf{else}\ \{Q\}$
			$\mid \mathbf{for}\ i = n\ \mathbf{to}\ m\ \{\ P\ \}$
			$\mid \mathbf{while}\ (e)\ \{P\} \mid A(p_1, \ldots, p_n)$
(*Actions*)	a	::=	$\mathbf{in}(T)@\ell \mid \mathbf{read}(T)@\ell \mid \mathbf{out}(t)@\ell$
			$\mid \mathbf{eval}(P)@\ell \mid x := e \mid \mathbf{inp}(T)@\ell$
			$\mid \mathbf{readp}(T)@\ell \mid \mathbf{rpl}(T) \rightarrow (t)@\ell$
			$\mid \mathbf{newloc}(s)$
(*Tuples*)	t	::=	$e \mid \ell \mid P \mid t_1, t_2$
(*Templates*)	T	::=	$e \mid \ell \mid P \mid !\,x \mid !\,l \mid !\,X$
			$\mid T_1, T_2$

Table 1: KLAIM syntax

processes and data tuples, can be allocated. KLAIM specifications consist of *nets*, namely finite plain collections of nodes where *components*, i.e. processes and data tuples, can be allocated. Nodes are composed by means of the parallel composition operator $\_ \parallel \_$. At net level, it is possible to restrict the visibility scope of a name s by using the operator $(\nu s)\_$: in a net of the form $N_1 \parallel (\nu s)N_2$, the effect of the operator is to make s invisible from within the subnet N_1.

Nodes have the form $s ::_\rho C$, where s is a unique locality name ρ is an allocation environment, and C is a set of hosted components. An *allocation environment* provides a name resolution mechanism by mapping locality variables l, occurring in the processes hosted in the corresponding node, into localities. The distinguished locality variable **self** is used by processes to refer to the address of their current hosting node. In the rest of this section, we will use ℓ to range over locality names and variables.

Processes are the KLAIM active computational units and may be ex-

ecuted concurrently either at the same locality or at different localities. They are built up from the process **nil**, which does nothing, and from basic actions by means of sequential composition $\_;\_$, parallel composition $\_ \mid \_$, conditional choice **if** (e) **then** $\{\_\}$ **else** $\{\_\}$, the iterative constructs **for** $i = n$ **to** m $\{\_\}$ and **while** (e) $\{\_\}$, and process definition $A(f_1, \ldots, f_n) \triangleq \_$.

During their execution, processes perform some basic *actions*. Actions $\mathbf{in}(T)@\ell$ and $\mathbf{read}(T)@\ell$ are retrieval actions and permit to withdraw/read *data tuples* (i.e. sequences of values) from the tuple space hosted at the (possibly remote) locality ℓ: if a matching tuple is found, one is non-deterministically chosen, otherwise the process is blocked. These actions exploit templates as patterns to select tuples in shared tuple spaces. *Templates* are sequences of actual and formal fields, where the latter are written $!x$, $!l$ or $!X$ and are used to bind variables to values, locality names or processes, respectively.

Actions $\mathbf{inp}(T)@\ell$ and $\mathbf{readp}(T)@\ell$ are non-blocking versions of the retrieval actions: namely, during their execution processes are never blocked. Indeed, if a matching tuple is found, **inp** and **readp** act similarly to **in** and **read**, and additionally return the boolean value *true*; otherwise, they return the value *false* and the executing process does not block. Actions $\mathbf{inp}(T)@\ell$ and $\mathbf{readp}(T)@\ell$ can be used where either a boolean expression or an action is expected (in the latter case, the returned value is simply ignored).

Action $\mathbf{out}(t)@\ell$ adds the tuple resulting from the evaluation of tuple t (which may contain expressions) to the tuple space of the target node identified by ℓ, while action $\mathbf{eval}(P)@\ell$ sends the process P for execution to the (possibly remote) node identified by ℓ. Actions **out** and **eval** are both non-blocking.

Action $\mathbf{rpl}(T) \rightarrow (t)@\ell$ atomically replaces a non-deterministically chosen tuple in ℓ matching the template T by the tuple t; if no tuple in ℓ matches T, the action behaves as $\mathbf{out}(t)@\ell$. Finally, action **newloc** creates new network nodes, while action $x := e$ assigns the value of e to x. These latter two actions, differently from all the others, are not indexed with an address because they always act locally.

2.4.2 Stochastic Analysis of KLAIM Specification

In this section we introduce the related analysis tool of KLAIM, that enables us to perform quantitative analysis of systems. In general, two main kind of analysis can be performed over systems, quantitative or qualitative analysis. In qualitative analysis we verify that a certain event will or will not occur. In quantitative analysis instead, we verify what is the probability that a certain event will or will not occur. In order to perform such analysis of KLAIM specification, we have to enrich such formalism by enabling the modelling of random phenomena. KLAIM specifications can be enriched with stochastic aspects, using the KLAIM's stochastic extension STOKLAIM, while the desired properties of the considered system can be expressed by using the stochastic logic MOSL. The properties of interest are then checked against the STOKLAIM specifications by means of the analysis tool SAM. In this section we provide an overview of STOKLAIM, MOSL and SAM.

STOKLAIM

In STOKLAIM [DKL$^+$06a; DLM05; DKL$^+$07], KLAIM's processes actions are enriched with a rate. Such rate is the parameter of an exponentially distributed random variable characterising the duration of the execution of an action. In particular, such random variables are governed by a negative exponential distribution. The negative exponential distribution is related to the Poisson distribution. It describe the times between events in a Poisson process, i.e. a process in which events occur continuously at a constant average rate λ; independently of the time t. A real valued random variable X has a *negative exponential distribution* with rate $\lambda > 0$ if and only if the probability that $X \leq t$, with $t > 0$, i.e. the probability that an event occurs within t time units, is $1 - e^{-\lambda \cdot t}$. The expected value of X is λ^{-1}, while its variance is λ^{-2}.

The operational semantics of STOKLAIM permits associating to each specification a Continuous Time Markov Chain (CTMC), one of the most popular models for the evaluation of the performance and dependability of information processing systems. Such CTMC is then used to perform

quantitative analyses of the considered system. The use of the exponential distribution is motivated by the fact that it enjoys convenient properties enabling automated analyses that are not always allowed by other distributions.

MoSL

The desired properties of a system under verification are formalised using the stochastic logic MoSL [DKL+07; DKL+06b]. MoSL formulae use predicates on the tuples located in the considered StoKlaim net to express the reachability of a certain system state, while passing through, or avoiding, other specific intermediate states.

The Mobile Stochastic Logic (MoSL) is an extension of a widely used temporal logic, CTL [EC82]. MoSL is inspired by (an action-based version of) CSL [ASSB00; BKH99], a stochastic extension of CTL that, together with qualitative properties, permits specifying time-bounded probabilistic reachability properties. The logic is both action- and state-based. MoSL incorporates some basic features of the Modal Logic for Mobility (MoMo) [DL05], in order to be able to refer to the distributed character of the specified systems. Specifically *basic state formulae* are built using a variant of the MoMo consumption ($\rightarrow$) and production ($\leftarrow$) operators. Intuitively, a consumption formula

$$A(p_1, ..., p_n)@\ell \rightarrow \Phi$$

holds for a network whenever in the network there exists a process A running at a node, of site ℓ, and the remaining network, namely $A(p_1, ..., p_n)$'s context, satisfies Φ. Notice that a process binder $!X$ can be used instead of process. Similarly formula

$$\langle T \rangle@\ell \rightarrow \Phi$$

holds whenever a tuple t matching T is stored in a node of site ℓ, and the remaining network satisfies Φ. The substitution resulting from pattern-matching is used to evaluate Φ.

A production formula

$$\langle t \rangle @\ell \leftarrow \Phi$$

holds if the network satisfies Φ whenever tuple t is stored in a node of existing site ℓ. In particular, the satisfaction of Φ is checked after the insertion of the tuple t in the network.

We can summarise the grammar for basic state formulae as follows:

$$\begin{aligned} \aleph ::= \quad & A(p_1, ..., p_n)@\ell \rightarrow \Phi \quad | \quad !X@\ell \rightarrow \Phi \quad | \quad < T > @\ell \rightarrow \Phi \\ | \quad & A(p_1, ..., p_n)@\ell \leftarrow \Phi \quad | \quad < t > @\ell \leftarrow \Phi \end{aligned}$$

MOSL distinguishes between *path* and *state* formulae. The basic format of *path formulae* is the CTL *until* formula $\Phi \mathcal{U} \Psi$. In particular, the logic use an action-based variant of the until operator, parameterised with two action sets: a path satisfies $\Phi \, {}_{\Delta}\mathcal{U}_{\Omega} \, \Psi$ whenever (eventually) a state satisfying Ψ (a Ψ-state) is reached via a Φ-path (i.e., a path composed only of Φ-states) and, in addition, while evolving between Φ-states, the performed actions satisfy Δ, and the Ψ-state is entered via an action satisfying Ω. Path formulae have also a time constraint. This is done by adding time parameter t which is either a real number or ∞. With time constraint, it is now imposed that a Ψ-state should be reached within t time units. Similarly, a path satisfies $\Phi \, {}_{\Delta}\mathcal{U}^{<t} \, \Psi$ if the initial state satisfies Ψ (at time 0) or eventually a Ψ state will be reached in the path, by time t via a Φ-path, and, in addition, while evolving between Φ-states, actions are performed satisfying Δ. Accordingly, the syntax of path formulae is:

$$\varphi ::= \quad \Phi \, {}_{\Delta}\mathcal{U}^{<t}_{\Omega} \, \Psi \quad | \quad \Phi \, {}_{\Delta}\mathcal{U}^{<t} \, \Psi.$$

State formulae are divided in three categories. The first category includes formulae in propositional logic, where the atomic propositions are tt and the basic state formulae $\aleph$ introduced previously in this section. The second category includes statements about the likelihood of paths satisfying a property, $\mathcal{P}_{\bowtie p}(\varphi)$. Finally the third category includes formulae for the so-called long-run properties, $\mathcal{S}_{\bowtie p}(\varphi)$. In general, a formula can be composed of sub-formulae of different categories. To be a

bit more precise about the probabilistic path properties. Let φ be a property imposed on paths. State s satisfies the property $\mathcal{P}_{\bowtie p}(\varphi)$ whenever the total probability mass for all paths starting in s that satisfy φ meets the bound $\bowtie\ p$. Here, $\bowtie$ is a binary comparison operator from the set $\{<, >, \leq, \geq\}$, and p a probability in $[0, 1]$. Long-run properties refer to the system when it has reached equilibrium. A state s satisfies $\mathcal{S}_{\bowtie p}(\varphi)$ if, when starting from s, the probability of reaching a state which satisfies Φ in the long run is $\bowtie p$.

In summary, state formulae are built according to the grammar:

$$\Phi, \Psi ::= \quad \mathsf{tt} \quad | \quad \aleph \quad | \quad \neg\Phi \quad | \quad \Phi \vee \Psi \quad | \quad \mathcal{P}_{\bowtie p}(\varphi) \quad | \quad \mathcal{S}_{\bowtie p}(\varphi)$$

SAM

Verification of MoSL formulae against StoKlaim specifications is assisted by the analysis tool SAM [DKL+07; Lor10], which uses a statistical model checking algorithm [CL10] to estimate the probability of the property satisfaction. In particular, the probability associated to a path formula is determined after a set of independent observations. Indeed, while in a numerical model checker the exact probability to satisfy a path formula is computed up to a precision, in a statistical model-checker the probability associated to a path formula is determined after a set of independent observations. This algorithm is parameterised with respect to a given tolerance ϵ and error probability p and guarantees that the difference between the computed values and the exact ones exceeds ϵ with a probability that is less than p.

2.4.3 KLAVA

KLAVA [BDP02] is a Java library providing the run-time support for KLAIM actions within Java code. This package relies on the IMC (Implementing Mobile Calculi) framework [BDF+05], which provides recurrent mechanisms for network applications and, hence, can be used as a middleware for the implementation of different formal languages. Specifi-

cally, KLAVA provides classes to be instantiated to create a net, and nodes that can be connected to the net in order to build the desired network environment. An abstract class is then provided to create processes to be added to the nodes, by instantiating subclasses specialized through inheritance and method overriding.

Tuples management is rendered in KLAVA by the class Tuple. Such class provides methods for creating a tuple, adding elements to a tuple, getting an element from a tuple, etc. A tuple can be created (Listing 2.1) by instantiating an empty tuple and then adding elements to it (using the method add(Object o)) or using one of the overloaded constructors.

Listing 2.1: Tuple Creation

```
1  //Empty tuple and adding elements
2  Tuple t1 = new Tuple();
3  t1.add(e1);
4  t1.add(e2);
5  t1.add(e3);
6
7  //Tuple creation by constructor
8  Tuple t2 = new Tuple(e1, e2, e3);
```

Nodes are implemented in KLAVA by the class KlavaNode and processes are implemented by the class KlavaProcess. KLAIM communication primitives are rendered in KLAVA by the following methods:

public void out(Tuple t, Locality l) **throws** KlavaException
public void read(Tuple t, Locality l) **throws** KlavaException
public void in(Tuple t, Locality l) **throws** KlavaException
public boolean read_nb(Tuple t, Locality l) **throws** KlavaException
public boolean in_nb(Tuple t, Locality l) **throws** KlavaException
public boolean read_t(Tuple t, Locality l, **long** TimeOut) **throws** KlavaException
public boolean in_t(Tuple t, Locality l, **long** TimeOut) **throws** KlavaException
public void eval(KlavaProcess P, Locality l) **throws** KlavaException

These methods take as parameters a tuple and the (either logical or physical) locality, i.e. the *address*, of the target node. If the action refers to the current execution site (through the reserved logical locality **self**),

it is simply redirected to the local tuple-space, otherwise a message will be sent to the remote target node. As expected, the method calls in(t,l), read(t,l) and out(t,l) are the implementation of the KLAIM actions $\mathbf{in}(T)@\ell$, $\mathbf{read}(T)@\ell$ and $\mathbf{out}(T)@\ell$, respectively. Instead the method calls in_nb(t,l) and read_nb(t,l) are the implementation of the KLAIM actions $\mathbf{inp}(T)@\ell$ and $\mathbf{readp}(T)@\ell$, respectively. Moreover in KLAVA we have two more method calls, namely in_t(t,l,time) and read_t(t,l,time). Such calls permit specifying upper bounds to the waiting time, i.e. a *time-out*, expressed in milliseconds. This is useful to deal with high network latency or absence of matching tuples.

To read all matching tuples only once in a loops, KLAVA provides specific built-in mechanisms that prevent matching twice the same tuple. In particular, method setHandleRetrieved() allows a tuple template to skip tuples already retrieved, while method resetOriginalTemplate() is used to reinitialize to empty values the formal fields[3] of a tuple template in order to use this template to retrieve another tuple in the next read_nb(t,l), read(t,l) or read_t(t,l,time) action.

Finally eval(P,l) spawns process P for execution at the remote site l.

In Chapters 4 and 5 we use KLAIM and its related tools for modeling, analysing and evaluating trust and reputation systems.

2.5 Probabilistic Trust

Here we briefly introduce a theoretical formalization of probabilistic trust and reputation systems, then we discuss some of the proposed models. Parties in a trust and reputation system are free to interact and rate each other. After each interaction, a rater assigns a score to a ratee. We denote by $\mathcal{R} = \{r_1, ..., r_m\}$ a finite, non-empty set of rating values. In probabilistic trust party's behaviour is modelled by a probability distribution on $\mathcal{R}$. Let $\Theta = \{\theta_1, \theta_2, ...\}$ be the set of possible parameters for a given distribution, we denote with $\mathcal{F} = \{p(\cdot|\theta)\}_{\theta \in \Theta}$ the set of probability

[3] A formal field is an item of a tuple subjects to substitution in case the match is satisfied. In KLAVA formal fields are distinguished from actual fields because they are created with the *default* constructor (i.e., the constructor with no parameters).

distributions on $\mathcal{R}$ indexed by parameters $\theta \in \Theta$. Let $r \in \mathcal{R}$ be a rating value and $p(\cdot \mid \theta)$ a distribution, the value $p(r \mid \theta)$ denotes the probability of observing a rating value r after an interaction with a party whose behaviour is (determined by) θ. The goal of a reputation system is to predict parties' behaviour in future interactions, given the rating values of past interactions. Thus, a reputation system has to provide the *reputation* of each party, i.e. an estimation $\tilde{\theta}$ of party's behaviour θ. The information about parties' past behaviour are denoted by a sequence of rating values $r^n = r_1, \ldots, r_n$, that is assumed to be a realization of a sequence of independent, identically distributed (i.i.d.) random variables $R^n = R_1, \ldots, R_n$. This assumption about i.i.d. random variables is common to all models we introduce below.

Beta Reputation System

Jøsang and Ismail [JI02] propose the *Beta reputation* system as a new system to foster good behaviour and to encourage adherence to contracts in e-commerce. The Beta reputation system is based on the Beta probability density function to derive parties' reputation values. The Beta distribution $Beta(\alpha, \beta)$, is defined over the interval $[0, 1]$ and is parametrized by two positive values, $\alpha > 0$ and $\beta > 0$. In this model the set $\mathcal{R}$ of rating values is the binary set $\{0, 1\}$, with values 0 and 1 denoting 'unsatisfactory' and 'satisfactory' interactions, respectively. Random variables R_i are assumed to be distributed according to a Bernoulli distribution with success probability $\theta \in [0, 1]$. It is thus assumed that when interacting with a party, whose behaviour is θ, the probability that the next interaction is 'satisfactory' is $p(1 \mid \theta) = \theta$, and 'unsatisfactory' is $p(0 \mid \theta) = 1 - \theta$.

Given the information of party's past interaction, the Beta reputation system compute the a posterior distribution over the parameter set Θ and set the party's reputation as the expected value of such distribution. Specifically, a Beta distribution $Beta(\alpha, \beta)$, where $\alpha = \beta = 1$, is used as a prior distribution over Θ. Thus the resulting a posterior distribution is a Beta distribution itself and parties' reputation is computed as follow: let α be the number of satisfactory past interactions with a party A and β the number of unsatisfactory interactions with A. The reputation of

party A is given by the expected value of a random variable ϑ distributed according to the Beta distribution $Beta(\alpha+1, \beta+1)$, defined as

$$\mathrm{E}[\vartheta] = \frac{\alpha+1}{\alpha+\beta+2} \qquad \alpha \geq 0\ , \beta \geq 0 \tag{2.11}$$

Jøsang and Ismail discuss also how to combine rating values from multiple sources and how to discriminate rating values from highly reputed parties giving more weight to them than rating values from parties with low reputation values. Finally they discuss the relevance of old rating values. They propose a forgetting scheme and the use of a forgetting factor in order to give less weight to old rating values than to more recent ones. In particular, the *forgetting factor* is a value in the interval $[0, 1]$. The aim of this parameter is to give a different weight to each rating based on its age. Let $r^n_{ord} = r_0, \ldots, r_n$ denote the ordered sequence (from the newest to the oldest) of rating values and λ be the forgetting factor. The weight associated to rating r_i is defined as λ^i, i.e. older ratings will be gradually forgotten. Thus, value $\lambda = 1$ is equivalent to absence of the forgetting factor, while value $\lambda = 0$ results in taking into account only the last rating (with the convention that $0^0 = 1$). The other possible values for λ approximate such extreme behaviours.

Dirichlet Reputation Systems

Jøsang and Haller in [JH07] propose an extension of the beta reputation system to the case of k discrete rating levels. We talk about Dirichlet reputation systems because for each value of k we have a different system. The reputation systems proposed are based on the Dirichlet probability distribution. The Dirichlet distribution $Dir(\alpha^k)$ is parametrized by a vector $\alpha^k = \alpha_1, ..., \alpha_k$ where parameters α_j, called *concentration parameters*, are such that $\alpha_j > 0$.

In these systems the set $\mathcal{R}$ of rating values is $\{r_1, ..., r_k\}$ and random variables R_i are assumed to be distributed according to a multinomial distribution of parameter $\vec{\theta} = \theta_1, ..., \theta_k$, with the constraint $\sum_{j=1}^{k} \theta_j = 1$. Thus, when interacting with a party whose behaviour is $\vec{\theta}$ the probability that the next interactions is rated r_j is $p(r_j|\vec{\theta}) = \theta_j$. The reputation system

computes party's reputation through the a posteriori distribution of parameter $\vec{\theta}$, given the information about party's past interactions. Party's reputation score is then given by the expected value of such distribution. Specifically, the a prior distribution over the parameters set is a Dirichlet distribution with concentration parameters $\alpha_j = \frac{2}{k}$, thus the resulting a posterior distribution is a Dirichlet distribution with concentration parameters $\alpha_j = n_j + \frac{2}{k}$, where n_j denotes the number of past interactions rated with r_j. The reputation of party A is given by the expected value of a random variable $\vec{\vartheta}$ distributed according to the Dirichlet distribution $Dir(\alpha^k)$ with $\alpha_j = n_j + \frac{2}{k}$, defined as follow

$$\mathrm{E}[\vartheta_j] = \frac{\alpha_i}{\sum_{j=1}^{k} \alpha_j} = \frac{n_j + 2/k}{2 + \sum_{j=1}^{k} n_j} \tag{2.12}$$

The choice of the a priori distribution is motivated by the authors as the necessarily choice in case of a uniform distribution for binary alternatives. Thus when $k > 2$ the a prior distribution is not anymore uniform. Another motivation is that, if a different distribution is chosen, in order to have an a priori uniform distribution, new ratings have less influence over the Dirichlet distribution, making the sensitivity to new evidence arbitrary small (for arbitrary values of k).

As in the case of Beta reputation system the authors discuss how to aggregate ratings and the aging of them through the use of a longevity factor. The longevity factor plays the same role of the forgetting factor in the Beta system, but it is defined differently.

ML Reputation Systems

We refer to [DA04] for the proposal of ML reputation systems, although the authors actually do not give a name to the systems. We refer to such systems as ML reputation systems because of the inference technique used to compute parties' reputation, namely the Maximum Likelihood (ML) estimation. Such estimation method tries at find the distribution's parameter θ that more likely produced the observed sequence of outcomes. In ML reputation systems, party's reputation is given by the pa-

rameter θ maximizing the likelihood,

$$L(\theta|R^n) = \mathbb{P}(R^n \mid \theta) = \prod_{i=1}^{n} \mathbb{P}(R_i = r_i \mid \theta), \tag{2.13}$$

where $\mathbb{P}$ denotes a probability function (see Section2.1).

It is often simpler to use the *log likelihood* $\mathcal{L} = \log L$, because of the product in 2.13 (indeed, the product $\prod_{i=1}^{n}$ becomes a summation, as reported in 2.14). The party's reputation then is written as

$$\tilde{\theta} = \arg\max_{\theta} \mathcal{L}(\theta|R^n) = \arg\max_{\theta} \sum_{i=1}^{n} \log \mathbb{P}(R_i = r_i \mid \theta). \tag{2.14}$$

In this reputation systems neither the set $\mathcal{R}$ of rating values nor the distribution of random variables R_i are fixed a priori. We briefly discuss here the case of Bernoulli distributions and categorical distributions.

Let the set $\mathcal{R}$ of rating values be the binary set $\{0, 1\}$ and random variables R_i be distributed according to a Bernoulli distribution. The party's reputation score $\tilde{\theta}$ of party A, namely the value θ maximazing $\mathcal{L}(\theta|R^n)$, is as follows

$$\tilde{\theta} = \frac{n_{satisfactory}}{n_{total}} \tag{2.15}$$

where $n_{satisfactory}$ denote the number of satisfactory interactions of A and n_{total} the total number of interactions of A.

In the case of categorical distributions the set $\mathcal{R}$ of rating values is the set $\{r_1, ..., r_k\}$ of cardinality k. Random variables R_i are distributed according to a categorical distribution of parameter $\vec{\theta} = \theta_1, ..., \theta_k$, with the constraint $\sum_{i=1}^{k} \theta_i = 1$. Thus the probability $\mathbb{P}(R = r_i|\vec{\theta}\,) = \theta_i$ denotes the probability that the next interaction with a party whose behaviour is $\vec{\theta}$ is rated with a value r_i. The party's reputation score of party A is computed as follows

$$\tilde{\theta}_i = \frac{n_i}{n_{total}} \qquad \text{for} \quad 1 \leq i \leq k \tag{2.16}$$

where n_i denotes the number of interactions in which A received a rating value r_i.

2.6 Online Systems

In this section we show some cases of reputation systems that are successfully used in real applications. We shortly describe the operation of the reputation systems implemented by eBay, Amazon and TipAdvisor.

eBay

eBay[4] is an online auction website in which users buy and sell a wide variety of goods. In eBay each user has associated a reputation score, representing the percentage of transactions in which the user has been considered trustworthy. Such percentage is computed on the basis of the transactions of the last twelve months. At the end of each transactions both the buyer and the seller have the opportunity to release a rating for the other party. When users release a rate they also write a short comment motivating their evaluation. Such comments are available to other users as complement of the reputation score. Each transaction can be evaluated with one of the following values: positive, negative or neutral. While positive and negative ratings are used for computing users' reputation, neutral ratings are just shown in the user profile, they do not influence the reputation, but it is possible to know their number for each user. Thus, users' reputation is computed as the percentage of positive interactions with respect to the total of interactions rated as positive and negative. For each auction in eBay it is possible to know the identity of the user that is offering the goods and her reputation. Such reputation system is used by eBay to encourage users, mainly sellers, to carry out transactions honestly without defraud buyers.

Amazon

Amazon[5] is an online retailer selling goods in several world countries. Amazon started as an online bookstore, but today is possible to find a large variaty of goods for sale. For each product sold in Amazon is asso-

[4]http://www.ebay.com/
[5]http://www.amazon.com/

ciated a reputation score, such score is a rational number between 1 and 5. Scores are denoted by stars, so a product can have associated a score between 1 and 5 stars. The ratings used for the computation of products' reputation are released by buyers who bought the product. Once a buyer have bought a product, she is encouraged to release a feedback for it, such feedback is composed by an integer value between 1 to 5 stars and a comment motivating the evaluation. All the ratings obtained by a product are then aggregated and used to compute the product's reputation, that is defined as the mean of all the rating values. Thus, for each product for sale in Amazon is shown the reputation score, the number of ratings used for the computation, and the comments released by buyers.

Amazon is not the only seller in its website, other sellers can sell their products through the Amazon website. Each seller has a reputation score that, as in the case of products' reputation, is a rational number between 1 and 5 stars. The ratings used for the computation of such value are released by buyers after a transaction with the seller. Sellers' reputation, as in the case of eBay, is computed on the basis of the transactions of the last twelve months. Moreover, as in the case of eBay, Amazon distinguishes among positive, negative and neutral feedbacks. Such distinction is made by dividing in three intervals the values between 1 and 5 stars: a positive feedback is a value between 4 and 5 stars, a neutral feedback is a value of 3 stars and a negative feedback is a value between 1 and 2 stars. The percentage for each seller of positive, neutral and negative feedback is reported in a table on the seller's profile webpage, together with the seller's reputation. Notice that in Amazon, as in eBay, ratings are released both by buyers and sellers, but unlike eBay the only visible ratings are the one released by buyers.

TripAdvisor

TripAdvisor[6] is a travel website that gather reviews from users about travel-related content. Users can review structures that are in one of the following categories: hotel, holiday rental, attraction, restaurant. For

[6]http://www.tripadvisor.com/

each review users have to release a score between 1 to 5 denoting respectively a terrible, poor, average, very good or excellent appreciation of the structure. Together with the rate, a comment and other few information about the trip and the structure are asked to users. The rating values are used to compute structures' reputation and are shown together with the comments written by users on the structures' profiles webpage. The structure reputation is the mean of the rating values received by users.

In TripAdvisor each user has a profile. Users' profiles contain personal information (whether the user uploaded her data) and a reference to each review the user released in TripAdvisor. Users have a kind of reputation score too, indeed for each user is reported the number of reviews that have been found useful by others. The idea is that the higher is such number the more trustworthy are the reviews of the user. One of the main criticism toward TripAdvisor is about the trustworthiness of users' reviews, because users can remain anonymous and there is not direct control on reviews' reliability. Thus, it could happen that, users review structures where they have not ever been or give high scores to structures because they know the owners. Users' reputation is used by TripAdvisor as means to tackle such phenomenon.

Chapter 3

A Theoretical Framework for Probabilistic Trust Systems

The potential usefulness and applicability of probabilistic trust is by now demonstrated by a variety of tools that have been experimentally tested in several contexts. There are very few *analytical* results on the behaviour of such systems – with the notable exception of the works by Sassone and collaborators [SKN07; KNS08]. Examples of questions that could be addressed by an analytical approach are: How do we quantify the *confidence* in the decisions calculated by the system? And how is this confidence related to such parameters as decision strategy and number of available ratings? Is there an optimal strategy that maximizes confidence as more and more information becomes available?

In this chapter, we address the above questions proposing a framework to analyze probabilistic trust systems based on *bayesian decision theory* [Rob07; Ber85; LJHG11]. A prominent aspect of this approach is the use of a priori probability distributions to model prior belief on the set of possible parties' behaviours. However, we also consider confidence measures that dispense with such prior belief. We study the behaviour of trust and reputation systems relying on the concept of *loss* function.

We quantify confidence in the system in terms of *risk* quantities based on *expected (a.k.a. bayes)* and *worst-case* loss. We study the behaviour of these quantities with respect to the available information, that is number of available rating values. Through the definition of reputation functions, and considering the way in which ratings are released, our results allow to characterize the asymptotic behaviour of probabilistic trust systems. We finally consider the case where the raters misbehave, that is exhibit a tendency to under- or over-estimate the quality of an interaction with a party. We argue that a data-model with *hidden variables* is well-suited to model this kind of scenario.

3.1 A Bayesian Framework for Trust and Reputation

In the following sections we discuss the main features of trust and reputation systems and we introduce our framework based on Bayesian decision theory for modelling such systems. The formal framework is composed by two main components: an *observation framework*, which describes how observations are probabilistically generated, and a *decision framework*, which describes how decisions are taken. Two essential ingredients of the latter are *loss* and *decision functions*. The following subsections are devoted to describing all of these elements.

3.1.1 Observation and Decision Framework

In trust and reputation system after each interaction, a rater assigns a score to a ratee. We denote by $\mathcal{O} = \{o_1, ..., o_m\}$ a finite, non-empty set of rating values: in the rest of the chapter we use the terms *outcomes*, *observables* and *rating values* interchangeably. We focus on trust and reputation systems where the behaviour/reputation of each party is modelled by a probability distribution on $\mathcal{O}$.

Definition 3.1.1 (Observation System) *An* observation system *is a quadruple* $\mathcal{S} = (\mathcal{O}, \Theta, \mathcal{F}, \pi(\cdot))$, *composed by a finite non-empty set of* observations $\mathcal{O}$, *a set of* world states *or* parameters Θ, *a set* $\mathcal{F} = \{p(\cdot|\theta)\}_{\theta \in \Theta}$ *of*

probability distributions on $\mathcal{O}$ indexed by Θ, and an a priori probability measure $\pi(\cdot)$ on Θ.

In a trust setting, the a priori measure $\pi(\cdot)$ expresses the user's belief over possible behaviours $\theta \in \Theta$ of the observed system. The data-model $\mathcal{F}$ represents how data are generated: the value $p(o|\theta)$ denotes the probability of observing a rating value $o \in \mathcal{O}$ in an interaction with a party whose behaviour is $\theta \in \Theta$.

Remark. The set Θ can be in principle discrete or continuous. In the following sections, unless otherwise stated, it is assumed that Θ is in fact *finite*. Moreover, we shall always assume the following conditions that simplify our treatment, with no significant loss of generality:

- for each θ, the distribution $p(\cdot|\theta)$ has full support, that is $p(o|\theta) > 0$ for each o. We shall sometimes denote the distribution $p(\cdot|\theta)$ as p_θ;
- for any pair of parameters $\theta \neq \theta'$, one has $p_\theta \neq p_{\theta'}$.

***Example* 3.1.1** A very simple possibility, but one widely used in practice, is to assume a set of binary outcomes representing *success* and *failure*, say $\mathcal{O} = \{o, \bar{o}\}$, which are generated according to a Bernoulli distribution: $p(o|\theta) = \theta$ and $p(\bar{o}|\theta) = 1 - \theta$, where $\theta \in \Theta \subseteq (0, 1)$.

Another possibility is to rate a service's quality by an integer value in a range of $n + 1$ values, $\mathcal{O} = \{0, 1, ..., n\}$. In this case, it is sometimes sensible to model the parties' behaviour by Binomial distribution $\mathcal{B}in(n, \theta)$, with $\theta \in \Theta \subseteq (0, 1)$ (somehow the discrete analog of a continuous Gaussian distribution). That is, the probability of an outcome $o \in \mathcal{O}$ for an interaction with a party with a behaviour θ is $p(o|\theta) = \binom{n}{o}\theta^o(1-\theta)^{n-o}$. In the following, we shall mostly concentrate on discretized sets of parameters. E.g. $\Theta = \{0.1, 0.2, \cdots, 0.9\}$.

3.1.2 Decision Framework

Reputation scores used to drive parties' interactions are computed on the basis of a party's past behaviour, given as a sequence of observations $o^n = (o_1, ..., o_n)$ ($n \geq 1$). These may derive from direct interaction of

the user, or be acquired by the user by different means (e.g. they may be gathered and provided by an online evaluation support system). Irrespective of how o^n is acquired, the basic idea – which will be studied analytically in Section 3.3 – is that the sequence o^n is a realization of a random vector $O^n = (O_1, ...O_n)$, where the random variables O_i are i.i.d. given $\theta \in \Theta$: $O_i \sim p(\cdot|\theta)$.

A decision $d = g(o^n) \in \mathcal{D}$ is taken on the basis of the past behaviour. This decision may however incur in a loss $\mathrm{L}(\theta, d)$, depending on the true behaviour θ of the ratee and on the taken decision itself. These concepts are formalized below.

Definition 3.1.2 (Decision Framework) *A* decision framework *is a quadruple* $\mathcal{DF} = (\mathcal{S}, \mathcal{D}, \mathrm{L}(\cdot,\cdot), \{g^{(n)}\}_{n\geq 1})$*, composed by an observation system* $\mathcal{S} = (\mathcal{O}, \Theta, \mathcal{F}, \pi(\cdot))$*, a decision set* $\mathcal{D}$*, a loss function* $\mathrm{L}(\cdot,\cdot) = \Theta \times \mathcal{D} \to \mathbb{R}^+$*, and a family of decision functions* $\{g^{(n)}\}_{n\geq 1}$*, one for each* $n \geq 1$*,* $g^{(n)} : \mathcal{O}^n \to \mathcal{D}$*.*

$\mathrm{L}(\theta, d)$ is a (in general, user-defined) function that quantifies the loss incurred when making a decision $d \in \mathcal{D}$, given that the real behaviour of the party is $\theta \in \Theta$. In the bayesian decision theory, the decision-making process is formalized via decision functions. For any n, a decision function is a function $g^{(n)} : \mathcal{O}^n \to \mathcal{D}$ (the superscript $^{(n)}$ will be normally omitted when n is clear from the context). We refer to Subsection 3.1.4 for examples and discussion about decision functions.

There are two main types of decisions one may wish to make when interacting with a party: evaluation of the party's behaviour, hence reputation; or the prediction of the outcome of the next interaction. In order to distinguish between these two cases, we define two instances of the above decision framework that differ by the definition of the decision set. In a *reputation framework*, one has $\mathcal{D} = \Theta$, that is the decision is made about the behaviour. In a *prediction* framework, one has $\mathcal{D} = \mathcal{O}$, that is the decision is made about the outcome of the (next) interaction.

3.1.3 Loss and Decision Functions

Loss functions (see Section 2.3) evaluate the consequences of possible decisions associating a loss to each decision. The choice of such functions depends on the application at hand and is, ultimately, responsibility of the user of the reputation system. For example, there could be monetary or economic losses connected to taking a given decision in a given state of the world. Below, we shall limit ourselves to indicate a few concrete examples of such loss functions.

For a reputation framework ($\mathcal{D} = \Theta$), one's objective may be to minimize a sort of distance between the true behaviour θ and the inferred reputation θ'. A common way to do so is to employ KL-divergence (see Section 2.2) as a measure of distance between probability distributions, and set: $\mathrm{L}(\theta, \theta') = D(p(\cdot|\theta')||p(\cdot|\theta))$. This loss function takes on a (proper) minimum value when $\theta = \theta'$, with $\mathrm{L}(\theta, \theta') = 0$. It is also a legitimate choice to consider the two arguments exchanged: $\mathrm{L}(\theta, \theta') = D(p(\cdot|\theta)||p(\cdot|\theta'))$, although the significance of this measure is less clear to us. Other measures can be based on distance between probability distributions seen as real valued vectors; one we shall consider is norm-1 distance: $\mathrm{L}(\theta, \theta') = ||p(\cdot|\theta) - p(\cdot|\theta')||_1$. Finally, if $\Theta \subseteq \mathbb{R}$, a sensible choice might be $\mathrm{L}(\theta, \theta') = |\theta - \theta'|$. For a prediction framework ($\mathcal{D} = \mathcal{O}$), one generally considers loss functions that are minimized when the probability of an outcome is maximum. One such loss function is $\mathrm{L}(\theta, o) = -\log p(o|\theta)$, that is, the Shannon information content of an outcome o: less probable the outcome o, more information/surprise (and loss) it conveys. Such function takes its minimum value for $o_\theta = \arg\max_{o \in \mathcal{O}} p(o|\theta)$ with $\mathrm{L}(\theta, o_\theta) = -\log \max_{o \in \mathcal{O}} p(o|\theta) = H_\infty(p_\theta)$. Here, H_∞ denotes *min-entropy* of a probability distribution/random variable.

3.1.4 Decision Functions

As already discussed, in trust and reputation systems the decision-making process consists of choosing a behaviour $\theta \in \Theta$ or a rating value $o \in \mathcal{O}$. We model such a process via decision functions. In this sec-

tion we discuss the definition of three specific decision functions. In the next section, we will argue that two of these functions are, in a precise sense, asymptotically optimal for all types of risks and of loss functions. It is possible to formulate decision functions through classical statistical inference procedures such as *Maximum Likelihood* (ML) and *Maximum A Posteriori* (MAP) estimation (see, e.g., [Hei]). Essentially, the ML rule yields the θ maximizing the likelihood of the observed o^n - equivalently, minimizing the KL distance between the empirical distribution of o^n and p_θ (see [Kul96]). The MAP rule yields the θ whose posterior probability given o^n is maximum.

Definition 3.1.3 (ML and MAP decision function) *Let $o^n = (o_1, ..., o_n)$ be a sequence of observations. Then a* ML *decision function $g^{(\mathtt{ML})} : \mathcal{O}^n \to \Theta$ satisfies*

$$g^{(\mathtt{ML})}(o^n) = \arg\min_\theta D(t_{o^n} || p(\cdot|\theta)).$$

A MAP *decision function $g^{(\mathtt{MAP})} : \mathcal{O}^n \to \Theta$ satisfies*

$$g^{(\mathtt{MAP})}(o^n) = \theta \;\; \textit{implies} \;\; p(\theta|o^n) \geq p(\theta'|o^n) \;\; \textit{for each} \;\; \theta' \in \Theta.$$

Note that implementation of the MAP rule implies knowledge of the a priori distribution $\pi(\cdot)$, which may be sometimes difficult or impossible to estimate. Fortunately, MAP and ML are asymptotically equivalent (and optimal). Despotovic and Aberer in [DA04] propose the use of reputation functions based on ML estimation; we use such functions in Section 3.3. Similarly, Jøsang and Ismail in [JI02] propose the use of a reputation function based on the Beta probability density function $Beta(\alpha, \beta)$. The reputation function is defined for a set of parameters $\Theta = [0, 1]$ and a binary set of ratings value $\mathcal{O} = \{o, \bar{o}\}$, representing satisfactory or unsatisfactory interactions.

Definition 3.1.4 (Beta decision function) *Let $o^n = (o_1, ..., o_n)$ be a sequence of observations and let $\mathcal{O} = \{o, \bar{o}\}$ be the observable set. Then the Beta decision function $g^{(\mathtt{Beta})} : \mathcal{O}^n \to \Theta = (0, 1)$ is defined as*

$$g^{(\mathtt{Beta})}(o^n) = \frac{\alpha + 1}{\alpha + \beta + 2}$$

where α is the number of occurrences of o in o^n and β is the number of occurrences of $\bar{o}$ in o^n.

3.2 Risk Analysis for Trust and Reputation Systems

In this section we introduce two measures for the evaluation of reputation functions, based on the expected and worst loss, respectively. We also discuss the notion of rate of convergence. In what follows, we fix a generic decision framework $\mathcal{S}$; for each $\theta \in \Theta$, we assume a decision $d_\theta \in \mathcal{D}$ exists that minimizes the loss given θ: $d_\theta = \arg\min_d \mathrm{L}(\theta, d)$. Let us first consider the definition of *frequentist risk*. For a parameter $\theta \in \Theta$, the frequentist risk associated to a decision function g after n observation is just the expected loss computed over $\mathcal{O}^n$, that is explicitly

$$R^n(\theta, g) = \sum_{o^n \in \mathcal{O}^n} p(o^n|\theta)\mathrm{L}(\theta, g(o^n)).$$

Relying on this definition we introduce first the bayes risk.

Definition 3.2.1 (Bayes risk) *Let $g : \mathcal{O}^n \to \mathcal{D}$ be a decision function and $\pi(\cdot)$ a prior probability distribution on the parameters set Θ. The* bayes risk *associated to g after n observations is the expected loss with respect to θ*

$$r^n(\pi, g) = \mathbb{E}_\pi[R^n(\theta, g)] = \sum_\theta \pi(\theta) R^n(\theta, g).$$

The minimum bayes risk *is defined as $r^* = \sum_\Theta \pi(\theta)\mathrm{L}(\theta, d_\theta)$.*

The bayes risk is the expected value of the risk $R^n(\theta, g)$, computed with respect to the a priori distribution $\pi(\cdot)$, that represents the a priori information over possible behaviours in the system. The second measure we introduce is the worst risk.

Definition 3.2.2 (Worst risk) *Let $g : \mathcal{O}^n \to \mathcal{D}$ be a decision function and Θ the parameters set. The* worst risk *associated to g after n observations is given by*

$$w^n(g) = \max_{\theta \in \Theta} R^n(\theta, g).$$

The minimum worst risk *is defined as $w^* = \max_{\theta \in \Theta} \mathrm{L}(\theta, d_\theta)$.*

The worst risk is thence the maximum risk $R^n(\theta, g)$ over possible parameters $\theta \in \Theta$.

***Example* 3.2.2** We compute the values of both minimum bayes and worst risks for two specific loss functions. We use the definitions of loss funcions given in previous sections. The first definition is for a reputation framework, the second for a prediction framework. Let $\mathrm{L}(\theta, \theta') = D(p(\cdot|\theta')||p(\cdot|\theta))$ be the loss function for a reputation framework. Then we have

$$r^* = \sum_{\Theta} \pi(\theta) D(p(\cdot|\theta)||p(\cdot|\theta)) = 0\,, \quad w^* = \max_{\theta\in\Theta} D(p(\cdot|\theta)||p(\cdot|\theta)) = 0.$$

Let now $\mathrm{L}(\theta, o) = -\log p(o|\theta)$ be the loss function for a prediction framework. We have

$$r^* = \sum_{\Theta} \pi(\theta) \log \frac{1}{p(o_\theta|\theta)} = \sum_{\Theta} \pi(\theta) H_\infty(p_\theta)\,,$$

$$w^* = \max_{\theta\in\Theta} \log \frac{1}{p(o_\theta|\theta)} = \max_{\theta\in\Theta} H_\infty(p_\theta).$$

3.3 Analysis Results

In this section we discuss some results about the convergence of risk quantities $r^n = r^n(\pi, g)$ and $w^n = w^n(g)$ to minimum bayes risk and worst risk, respectively, and their rates of convergence. We first examine risks in a reputation and in a decision framework; then briefly discuss exponential bounds on the probability of exceeding a given loss.

Given a decision framework, it is important to establish not only the *limit* of the risk functions, r^n and w^n, as the number n of available ratings grows, but also *how fast* this limit is approached. The *rate* of convergence tells us how fast this limit is approached. The concept of rate is important for two reasons. Firstly, it is desirable to distinguish between reputation functions with different rates, as a reputation function with a high rate may require considerably less observations in order to achieve an improvement of the risks value, compared to a reputation function with a low rate. Secondly, knowledge of the rate will allow us to obtain quick and accurate estimations of the risk functions r^n and w^n depending on n.

3.3.1 Reputation

In what follows, for the purposes of our analysis, we fix a generic reputation framework $\mathcal{RF} = (\mathcal{S}, \Theta, \mathrm{L}(\cdot,\cdot), \{g^{(n)}\}_{n\geq 1})$. In order to discuss our results in the simplest possible form, we shall assume Θ is finite, and that $\mathrm{L}(\theta,\theta') > \mathrm{L}(\theta,\theta)$ for each $\theta \neq \theta'$. We let

$$R = \min_{\theta\neq\theta'} C(p_\theta, p_{\theta'})$$

be the least Chernoff Information between any pair of distinct distributions p_θ and $p_{\theta'}$ in $\mathcal{F} = \{p(\cdot|\theta) \,|\, \theta \in \Theta\}$. We shall often abbreviate $r^n(\pi, g)$ as just r^n, similarly for w^n.

The following theorem states that the best achievable rate of convergence to the minimum values of any decision function, for both bayes and worst risks, is bounded above by R.

Theorem 3.3.1 *Assume* $\lim r^n(\pi,g)$ *exists. Then*

1. $\lim_{n\to\infty} r^n(\pi,g) \geq r^*$;
2. *if* $\lim_{n\to\infty} r^n(\pi,g) = r^*$ *then* $\mathrm{rate}(r^n(\pi,g)) \leq R$, *if defined.*

Similarly for the worst risks w^n *and* w^*.

Proof. We first consider the case of bayes risk r^n and r^*. For each $n \geq 1$ and θ, let $A_\theta^{(n)} = {g^{(n)}}^{-1}(\theta)$ denote the acceptance region for θ. We start by considering the frequentist risk and obtain the following chain of (in)equalities:

$$\begin{aligned}
R^n(\theta,g) &= \sum_{\mathcal{O}^n} p(o^n|\theta)\mathrm{L}(\theta, g(o^n)) && (3.1)\\
&= \sum_{\theta'\neq\theta} p(A_{\theta'}^{(n)}|\theta)\mathrm{L}(\theta,\theta') + p(A_\theta^{(n)}|\theta)\mathrm{L}(\theta,\theta) && (3.2)\\
&= \sum_{\theta'\neq\theta} p(A_{\theta'}^{(n)}|\theta)\mathrm{L}(\theta,\theta') + (1 - p({A_\theta^{(n)}}^c|\theta))\mathrm{L}(\theta,\theta) && (3.3)\\
&= \sum_{\theta'\neq\theta} p(A_{\theta'}^{(n)}|\theta)\mathrm{L}(\theta,\theta') + (1 - \sum_{\theta'\neq\theta} p(A_{\theta'}^{(n)}|\theta))\mathrm{L}(\theta,\theta) && (3.4)\\
&= \mathrm{L}(\theta,\theta) + \sum_{\theta'\neq\theta} p(A_{\theta'}^{(n)}|\theta)(\mathrm{L}(\theta,\theta') - \mathrm{L}(\theta,\theta)) && (3.5)\\
&\geq \mathrm{L}(\theta,\theta) && (3.6)
\end{aligned}$$

where the last inequality stems from $\mathrm{L}(\theta,\theta') > \mathrm{L}(\theta,\theta)$. Averaging on all θ's we obtain the first part.

Concerning the second part, let $m = \min_{\theta\neq\theta'} \mathrm{L}(\theta,\theta') - \mathrm{L}(\theta,\theta)$, where $m > 0$. From (3.5), we get

$$R^n(\theta, g) \geq \mathrm{L}(\theta,\theta) + p(A_\theta^{(n)^c}|\theta) \cdot m$$

which, when averaged over all θ's, and taking into account that $P_e^{(g)}(n) = \sum_\theta \pi(\theta) p(A_\theta^{(n)^c}|\theta)$, yields

$$r^n(\pi, g) \;\geq\; r^* + P_e^{(g)}(n)m \;\geq\; r^* + P_e(n)m$$

where $P_e(n)$ denotes the MAP probability of error and the last step stems from the optimality of MAP. The above inequality implies $-\frac{1}{n}\log(r^n - r^*) \leq -\frac{1}{n}\log(P_e(n)m)$. Taking the limit on both sides (assuming it exists on the left-hand side), from (2.10) we obtain that the rate of r^n, if it exists, is $\leq \mathrm{rate}(P_e(n)) = R$.

The case of the worst risk w^n is similar, as the rate of $r^n = \sum_\theta R^n(\theta, g)$ is determined by the slowest of the summands $R^n(\theta, g)$, that is the one with the smallest rate, which also yields $\max_\theta R^n(\theta, g)$, for n large enough. ∎

The following theorem confirms that both MAP and ML are asymptotically optimal decision functions. Such functions achieve minimum loss value and maximum rate of convergence.

Theorem 3.3.2 *Assume g is either a* MAP *or a* ML *decision function. Then $\lim_{n\to\infty} r^n = r^*$ and moreover $\mathrm{rate}(r^n) = R$. Similarly for w^n.*

Proof. We consider the case of bayes risk r^n. Let g be a MAP function first. Let $M = \max_{\theta\neq\theta'} \mathrm{L}(\theta,\theta') - \mathrm{L}(\theta,\theta) > 0$. From (3.5), it is easy to prove that

$$\begin{aligned} R^n(\theta, g) &\leq \mathrm{L}(\theta,\theta) + \sum_{\theta'\neq\theta} p(A_{\theta'}^{(g)}|\theta) \cdot M & (3.7)\\ &= \mathrm{L}(\theta,\theta) + p(A_\theta^{(g)^c}|\theta) \cdot M\,. & (3.8) \end{aligned}$$

Averaging over θ, this implies

$$r^n \leq r^* + P_e(n) \cdot M \tag{3.9}$$

where as usual $P_e(n)$ denotes the MAP error probability. From Theorem 3.3.1(1) we know that it must be $r^n \geq r^*$, hence $\lim_n r^n = r^*$, which proves the first part. Concerning the rate, (3.9) implies that $\mathrm{rate}(r^n) \geq \mathrm{rate}(P_e(n)) = R$; the thesis for this part then follows from Theorem 3.3.1(2).

The case of g ML is equivalent to the case of g MAP with a uniform prior over Θ, which yields again the same results.

For the case of the worst risk w^n, the same considerations discussed in the proof of Theorem 3.3.1 apply. ∎

In essence, the above results can be summarized by saying that, under an optimal decision function, r^n behaves as $\approx r^* + 2^{-nR}$, and w^n as $\approx w^* + 2^{-nR}$.

3.3.2 Prediction

We fix a generic prediction framework $\mathcal{PF} = (\mathcal{S}, \mathcal{O}, \mathrm{L}(\cdot,\cdot), \{h^{(n)}\}_{n\geq 1})$. For a distribution p, let $\mathrm{Argmax}(p)$ denote the set of observables maximizing $p(o)$. We assume Θ is finite, and that for each θ there is $o_\theta \in \mathrm{Argmax}(p_\theta)$ such that $\mathrm{L}(\theta, o) > \mathrm{L}(\theta, o_\theta)$ for each $o \neq o_\theta$. Like in the preceding subsection, we let R denote the minimum Chernoff Information between any pair of distinct distributions on $\mathcal{O}$.

We shall limit our discussion to the following result of practical interest, which describes the form of the (optimal) prediction function: as expected, given o^n, one has to first identify the underlying distribution p_θ and then take the observable that maximizes this distribution. The proof goes along the lines of the results in the preceding subsection and is omitted.

Theorem 3.3.3 *Let $g = \{g^{(n)}\}_{n\geq 1}$ be a family of* ML *decision functions. Assume $h^{(n)}(o^n) = o_\theta$ where $\theta = g^{(n)}(o^n)$. Then $\lim_{n\to\infty} r^n = r^*$ and $\mathrm{rate}(r^n) = R$. Similarly for w^n.*

3.3.3 More Exponential Bounds

We discuss here the probability that the loss deviates from its minimal value above a given threshold. These results apply to the case g is the ML or MAP decision functions. Such results do not depend on the parameter θ but only on the number n of observations and the threshold value fixed for the loss. The results also apply to the case when Θ is continuous. We begin by considering the KL-loss function.

Theorem 3.3.4 *Let g be the* ML *or* MAP *decision function. Let $\mathrm{L}(\theta, \theta') = D(p(\cdot|\theta')||p(\cdot|\theta))$ be the KL loss function. Fix any $\theta \in \Theta$ and assume O^n is a n-sequence of random variables i.i.d. given θ, that is $O_i \sim p(\cdot|\theta)$. Let $\epsilon > 0$. Then*

$$\Pr(\mathrm{L}(\theta, g(O^n)) > \epsilon) \leq (n+1)^{|\mathcal{O}|} 2^{-n\epsilon}\,.$$

Proof. Consider the set $U_\theta^{(n)}(\epsilon) = \{o^n \in \mathcal{O}^n | D(t_{o^n}||p(\cdot|\theta)) \leq \epsilon\}$. By definition of g and $\mathrm{L}(\theta, \theta') = D(p(\cdot|\theta')||p(\cdot|\theta))$ we have

$$\begin{aligned} \Pr(\mathrm{L}(\theta, g(O^n)) > \epsilon) &= \Pr(O^n \notin U_\theta^{(n)}(\epsilon)) \\ &= \Pr(D(t_{O^n}||p(\cdot|\theta)) > \epsilon) \leq (n+1)^{|\mathcal{O}|} 2^{-n\epsilon} \end{aligned}$$

where the last inequality follows by Theorem 11.2.1 in [CT06]. ∎

The above result can be easily extended to the case of norm-1 loss function.

Corollary 3.3.5 *Let g be the* ML *or* MAP *decision function. Let $\mathrm{L}(\theta, \theta') = ||p(\cdot|\theta') - p(\cdot|\theta)||_1$, be the norm-1 distance loss function. Fix any $\theta \in \Theta$ and assume O^n is a n-sequence of random variables i.i.d. given θ, that is $O_i \sim p(\cdot|\theta)$. Let $\gamma > 0$. Then*

$$\Pr(\mathrm{L}(\theta, g(O^n)) > \gamma\ ) \leq (n+1)^{|\mathcal{O}|} 2^{-n\frac{\gamma^2}{2\ln 2}}\,.$$

Proof. A consequence of the preceding theorem and of Pinsker's inequality (Lemma 11.6.1 in [CT06]), which relates norm-1 distance to KL-divergence: for any two distributions p and q on the same finite set $\mathcal{O}$, we have $D(p||q) \geq \frac{1}{2\ln 2}||p-q||_1^2$. ∎

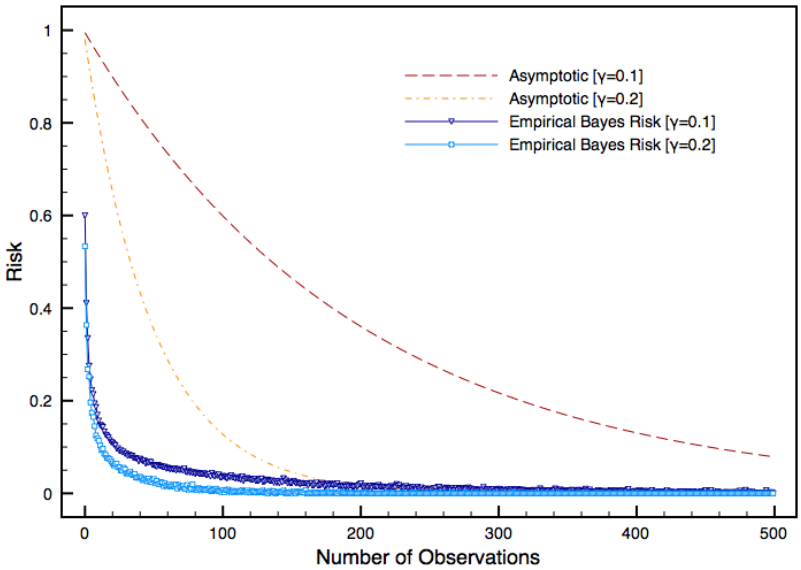


Figure 2: Empirical and asymptotic bayes risk trend for $\gamma = 0.1$ and $\gamma = 0.2$

3.4 Examples of Systems Assessment

We simulate a system composed by a number of peers, for our study we used the GNU Octave[1] software. In our first experiment, the behaviour of each peer is modelled as a Bernoulli distribution $\mathcal{B}(\theta)$ over the set $\mathcal{O} = \{0, 1\}$, representing successful and unsuccessful interactions, with θ being the success probability. The parameter θ is drawn from a fixed, finite set $\Theta \subseteq (0, 1)$: we assume Θ to be a discrete set of N points $0 < \gamma, 2\gamma, ..., N\gamma < 1$, for a fixed positive parameter γ. Moreover, we assume a uniform a priori distribution $\pi(\cdot)$ over Θ. We consider the case of reputation, and fix the loss function to be $\mathrm{L}(\theta, \theta') = ||p(\cdot|\theta) - p(\cdot|\theta')||_1$. The purpose of this first experiment is twofold: (a) to study the rate of convergence of the risk functions to their limit values depending on γ;

[1] http://www.gnu.org/software/octave/

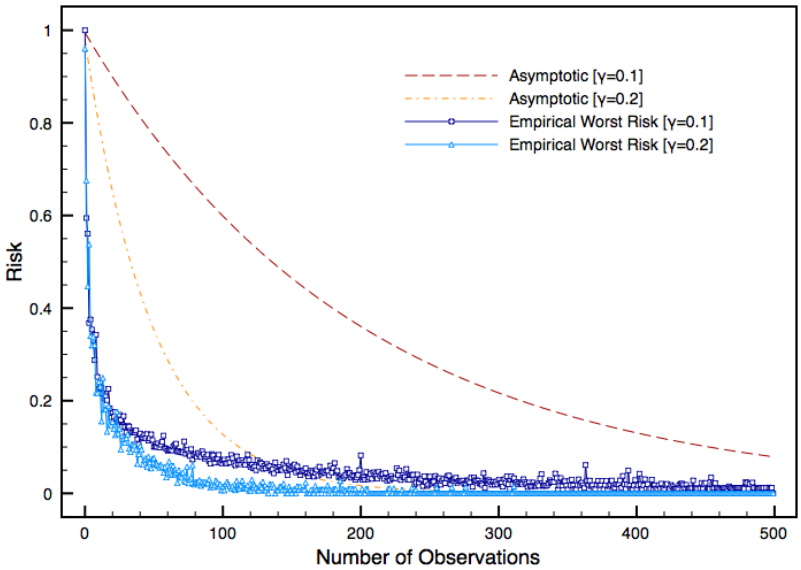


Figure 3: Empirical and asymptotic worst risk trend for $\gamma = 0.1$ and $\gamma = 0.2$

and (b) to compare the analytical approximations

$$r^n \approx r^* + 2^{-nR} \text{ and } w^n \approx w^* + 2^{-nR},$$

where $R = \min_{\theta \neq \theta'} C(p_\theta, p_{\theta'})$, with the empirical values of r^n and w^n obtained through the simulations.

For our analysis a ML reputation function g is used: we know by Theorem 3.3.2 that the convergence to both minimum bayes and worst risks is assured for this kind of functions. Varying the value of γ, we analyse how the rate of convergence changes. We consider the following values: $\gamma = 0.2$, $\gamma = 0.1$ and $\gamma = 0.05$, for which we get the values of the rate: 0.029447, 0.0073242 and 0.0018102, respectively. For smaller values of γ an improvement on the risk values requires a larger amount of observations, compared to larger values of γ. Figures 2 and 3 graphically show the trend of bayes and worst risks with respect to different values of γ.We compare the asymptotic approximation of the bayes and worst risks with their empirical values obtained through simulations, for different values

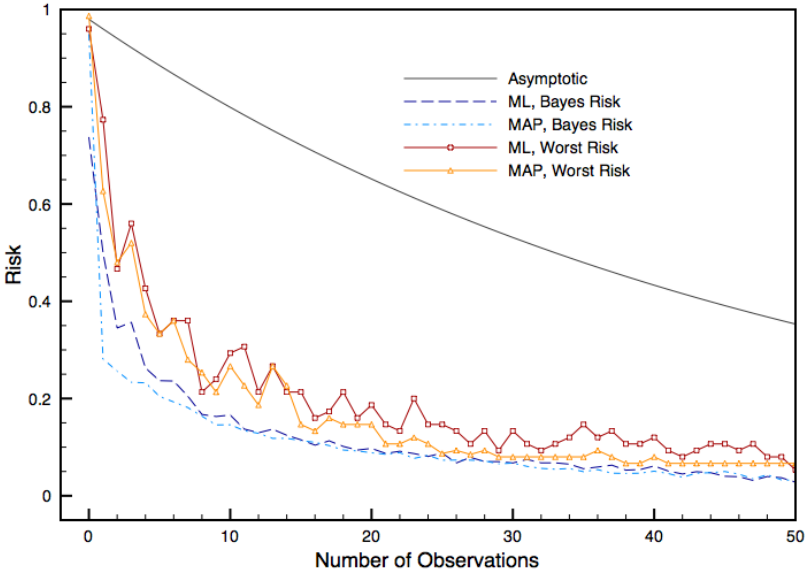


Figure 4: Bayes and worst risks trend for different reputation functions

of γ. The value of γ affects the risk value because it determines the structure of the set Θ. Intuitively what happens is that: for large values of γ, the incurred loss will be exactly zero in most cases. On the contrary, for small values of γ, the incurred loss will be small but nonzero in most cases. As we can observe in Figures 2 and 3 , the bayes and worst risks converge to their limit values according to their rates of convergence. Moreover, there is a good agreement between the asymptotic approximation and the empirical values.

In our second experiment, we analyse a system with respect to the use of different reputation functions. Like in the previous example, each peer exhibits a Bernoulli behavior, with the parameter θ ranging in Θ. This time, though, the prior distribution $\pi(\cdot)$ on Θ is a binomial centered on the value $\theta = 0.5$, $\mathcal{B}in(|\Theta|, 0.5)$. We compare the use of a ML and a MAP reputation function, through the study of both bayes and worst risks trend. For simplicity, we fix the value of γ to 0.2 and we

use norm-1 distance as a loss function. Figure 4 shows the trend of both bayes and worst risks for both reputation functions. As expected, MAP performs significantly better than ML when a small number of observations is available. This difference is due to the the fact that MAP actually takes advantage of the a priori knowledge represented by $\pi(\cdot)$, differently from ML. As the number of observations increases the effect of the prior washes out and both functions converge to the same value – r^* or w^*, depending on the chosen risk model – as predicted by the theory.

3.5 A refined rating mechanism

In this section we discuss a refinement of the observation and rating mechanism that takes into account possible raters' misbehaviour. In certain environments, it might be the case that not all the parties in the system will rate faithfully others parties: some raters may have a (possibly malicious) tendency to under- or over-evaluate the outcomes of interactions they are involved in. To take into account such form of misbehaviour, we have to consider a refined data generation model, where the probability of observing a given rating value does not depend solely on the behaviour of the ratee, but also on some *unobservable* bias characterizing the rater's behaviour.

3.5.1 Partial Observation Model

In this refined model, we assume an additional nonempty set $\mathcal{H}$ of *raters behaviours*, over which an a priori probability measure $\rho(\cdot)$ is given. In general, the behaviour of the system is characterized by a joint probability measure $p(o, h, \theta)$. However, as a simplifying assumption, we postulate that the rater's behaviour h is independent of the ratee's behaviour θ. In other words, we postulate the following factorization:

$$p(o, h|\theta) = p(o|h, \theta)\rho(h) \tag{3.10}$$

for each o and θ (in case the variables h and θ were not independent, in the expression above one should replace $\rho(h) = p(h)$ by $p(h|\theta)$). This

means that the data generation model is completely specified by a table $p(o|h,\theta)$ plus the priors π and ρ, as stated in the following definition.

Definition 3.5.1 (Partial Observation System) *A* partial observation system *is a tuple* $\mathcal{S} = (\mathcal{H}, \mathcal{O}, \Theta, \mathcal{F}, \pi(\cdot), \rho(\cdot))$, *where* $\pi(\cdot)$ *and* $\rho(\cdot)$ *are probability measures on* Θ *and* $\mathcal{H}$, *respectively, and* $\mathcal{F} = \{p(\cdot|h,\theta)\}_{h\in\mathcal{H},\theta\in\Theta}$ *is a parametric family of probability distributions on* $\mathcal{O}$.

We discuss below some examples of partial observation systems.

***Example* 3.5.3** Consider a system with two types of raters, honest and dishonest, two types of ratings, satisfactory and unsatisfactory and where the behaviour of each ratee is modelled by a Bernoulli distribution. More precisely, we let $\mathcal{H} = \{0, 1\}$ be the set of raters behaviours, where $h = 0$ denotes an honest rater and $h = 1$ a dishonest one. We let $\mathcal{O} = \{0, 1\}$ be the set of rating values, where $o = 0$ and $o = 1$ denote unsatisfactory and satisfactory interactions, respectively. We assume a set $\Theta \subseteq (0, 1)$ of ratees' parameters. Assuming that dishonest raters give a rating value stating the opposite of the actual interaction outcome, while honest raters do not modify it, we model the rating mechanism $p(o|h,\theta)$ as follows: $p(\cdot|h = 0,\theta) = \mathcal{B}(\theta)$ and $p(\cdot|h = 1,\theta) = \mathcal{B}(1-\theta)$.

***Example* 3.5.4** Consider a system where each rating values are integers in $\mathcal{O} = \{0, 1, ..., m\}$. We assume that the behaviour of each ratee is modelled by a certain binomial distributions $\mathcal{B}in(m,\theta)$, with $\theta \in \Theta \subseteq (0, 1)$. Assume now that raters are partitioned into three types: pessimistic, optimistic and neutral raters. Specifically, we consider a set of raters' behaviour $\mathcal{H} \subseteq (-1, 1)$, where $h < 0$ denotes a pessimistic rater, $h > 0$ an optimistic one and $h = 0$ a neutral rater. In such a setting, we model the rating mechanism $p(o|h,\theta)$ as follows: $p(\cdot|h,\theta) = \mathcal{B}in(m,\theta(1+h))$ if $h < 0$, $p(\cdot|h,\theta) = \mathcal{B}in(m,\theta)$ if $h = 0$, and $p(\cdot|h,\theta) = \mathcal{B}in(m,[\theta+(1-\theta)h])$ if $h > 0$.

***Example* 3.5.5** Consider a system with two types of ratings, satisfactory and unsatisfactory and where the behaviour of each ratee is modelled by a Bernoulli distribution. Thus $\mathcal{O} = \{0, 1\}$. We assume that raters

are grouped into three types, exhibiting different bias about the ratees's behaviour.

Informally, raters of the first type tend, at varying degrees, to *amplify* their perception of the ratees' behaviour. Thus, a ratee that behaves mostly satisfactorily will have, with this type of raters, higher probability of receiving positive evaluations, compared to her actual behaviour. Similarly, a ratee that behaves mostly unsatisfactorily will have, with this type of raters, higher probability of receiving negative evaluations, compared to her actual behaviour. Raters of the second type are pathological liars that tend, at varying degrees, to amplify *at the opposite* their perception of the ratees' behaviour: ratees acting positively will have a greater probability of receiving negative evaluations, and vice-versa. Finally, neutral raters are objective evaluators, that do not exhibit neither types of bias.

Formally, we consider the set of raters' behaviour $\mathcal{H} = \mathbb{R}$, where $h > 0$ denotes a rater of the first type, $h < 0$ a rater of the second type and $h = 0$ is an objective rater. We model the rating mechanism $p(o|h, \theta)$ by a form of exponential distribution (specifically, a Gibb's distribution):

$$p(o = 1|h, \theta) = \frac{\theta e^{h\theta}}{Z} \qquad p(o = 0|h, \theta) = \frac{(1-\theta)e^{h(1-\theta)}}{Z}$$

$$\text{where } Z = \theta e^{h\theta} + (1 - \theta)e^{h(1-\theta)} \quad .$$

Note that, as $h \to +\infty$, for $\theta > 0.5$, we have $p(o = 1|h, \theta) \to 1$ while for $\theta < 0.5$ we have $p(o = 1|h, \theta) \to 0$. The situation for $h \to -\infty$ is specular. If the behaviour of the ratee is the Bernoulli distribution of parameter $\theta = 0.5$, i.e. the values are equiprobable, the raters' behaviour does not have any effect on the evaluation.

3.5.2 Computing Decision Functions

We briefly discuss the computation of the decision functions when the rating mechanism introduced in the previous section is assumed. First, we make the data-generation model precise in the case of multiple observations. We assume that the observations $o^n = o_1, ..., o_n$ and the corresponding hidden raters' values $h^n = h_1, ..., h_n$ are generated i.i.d.

given the parameter $\theta \in \Theta$. In other words, we assume a sequence of pairs of random variables $(O_1, H_1), (O_2, H_2), ...$ that are i.i.d. and such that $(O_i, H_i) \sim p(o, h|\theta)$, for $\theta \in \Theta$. In our model, the rater's parameter h is however *never* observed: according to the statistical terminology, h is *missing completely at random*. Under this assumption, we can use the marginal likelihood of the observable parameters, $p(o^n|\theta)$, to estimate the parameter θ (see, e.g., [Bar12]). In other words, both ML and MAP, seen as functions of the visible parameter o^n, are still valid decision functions, and the results shown in Section 3.3 carry over immediately to the present setting.

Now, let us discuss how a ML decision function could be efficiently implemented – the discussion for MAP is conceptually similar and omitted. Implementing ML is equivalent to maximizing $p(o^n|\theta)$ w.r.t. to θ, given o^n. In principle, taking (3.10) into account, for any given θ one can compute $p(o^n|\theta)$ from the data available in the model by ordinary marginalization, as follows (below, $p(o^n|h^n, \theta) = \Pi_i p(o_i|h_i, \theta)$; the extension of measure ρ to $\mathcal{H}^n$ is still denoted by ρ; integrals are to be replaced by ordinary summations in case $\mathcal{H}$ is discrete):

$$p(o^n|\theta) \;=\; \int_{\mathcal{H}^n} p(o^n, h^n|\theta)\, d\rho(h^n) \;=\; \int_{\mathcal{H}^n} p(o^n|h^n, \theta)\rho(h^n)\, d\rho(h^n)\,. \tag{3.11}$$

In practice, the rightmost term in (3.11) can be difficult to compute, let alone maximize: in particular, the hidden variables h^n introduce a dependency between $p(o^n|h^n, \theta)$ and $\rho(h^n)$. Fortunately, there is a well known approach for the practical computation of the Maximum Likelihood in the presence of hidden variables, the *Expectation-Maximization (*EM*)* algorithm [Bar12]. The EM algorithm is an iterative procedure used in the cases where the equations for the maximization of the likelihood cannot be solved directly. Such an approach can be used for efficiently implementing the decision functions we have considered in the previous sections[2]. We refer to the relevant literature for further details, e.g. [Bar12].

[2]To take effectively advantage of EM, one may have to impose restrictions on the form of Θ (typically assumed to be continuous) and $\mathcal{H}$ (typically assumed to be discrete).

Chapter 4

Analysis of Reputation Systems Specifications

Once a reputation system has to be deployed in a network environment and set up for such environment, questions like the following may arise: Which reputation model is more suitable for the given environment? Does the model meet the expected behaviour? How does parties' behaviour affect their reputations? How do their initial reputation scores affect the model?

In this chapter, we propose using a *coordination language* equipped with a formal semantics to deal with the above issues. On the one hand, in the last two decades coordination languages have been used for modeling and programming a variety of different systems that are operating in open and non-deterministic environments [OV11]. In particular, tuple-based languages (see, e.g., [RCD01] for a survey) have been effectively used to implement coordination mechanisms in a distributed setting. On the other hand, formal methods are perfectly suitable means to precisely describe the relevant aspects of distributed systems, to state and prove their properties, and to direct attention towards issues that might otherwise be overlooked. For our analysis we focus on the coordination language KLAIM [DFP98], because it provides powerful coordination mechanisms based on the tuple-based communication model

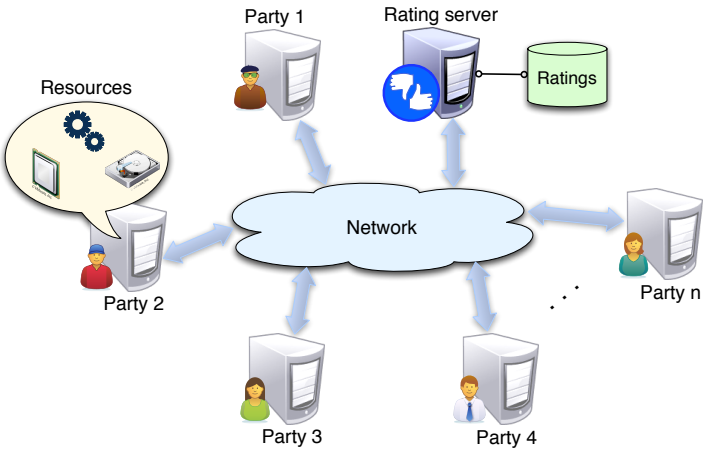


Figure 5: Networked Trust Infrastructure

and, moreover, it is equipped with formal tools supporting verification and, most importantly, it has been specifically designed for network-aware modelling of distributed applications. Thus, KLAIM appears to be well suited for specifying trust and reputation systems and for reasoning about them.

A *networked trust infrastructure* allows parties of a reputation system to exchange ratings and interact. For our analysis we consider a general infrastructure, graphically depicted in Figure 5, where a *rating server* collects all ratings from system's parties and makes them publicly available. Every *party* can play the role of a client, of a provider, or both, and may offer different kinds of resources (e.g., CPU time, disk space, files, services). Therefore, whenever a party needs a resource, it queries the rating server to determine the reputation of all parties providing the resource. Then, it selects one of the providers with the highest reputation score and, after the interaction, rates it according to the quality of the provided resource. Notably, we consider a centralised rating server, because it is a widely used setting for networked trust infrastructures and how ratings are stored in the network is not a parameter of interest at this

level of analysis. On top of the general infrastructure described so far, different kinds of reputation system can be layered, which mainly differ for the model they use to aggregate ratings when computing reputation scores.

In our analysis we proceed in three steps as follows:

1. We model the considered reputation system with KLAIM. Specifically, we provide a 'schema' specification that is parametric w.r.t. the reputation model used to determine the parties' reputation (and w.r.t. other parameters of the system). The specification of the given system is obtained by appropriately instantiating the parameters of the generic specification.

2. We enrich the specification with stochastic aspects, using the KLAIM's stochastic extension STOKLAIM [DKL+07], and formally express the desired properties using the stochastic logic MOSL [DKL+07].

3. We check the properties of interest against the STOKLAIM specification by means of the analysis tool SAM [DKL+07; Lor10].

We remind the reader that KLAIM and related analysis tools are described in Section 2.4. We also remind that, in our work, we focus on reputation models that use a probabilistic approach for computing reputation scores. The verification methodology proposed in this chapter is aimed at giving a contribution to the design and integration issues of reputation systems. Specifically our proposal is meant to be used in the design phase, i.e. when developers have to choose which reputation system is more suitable for the given environment and have to tune system parameters.

4.1 Formal Specification of Reputation Systems

In this section, we present the relevant aspects of the KLAIM specification of a reputation system deployed in the general infrastructure previously

described and graphically depicted in Figure 5. For the sake of readability, the specification code reported in this section is sometimes accompanied by comments (strings preceded by // indicates that the rest of the line is a comment).

The overall system can be rendered in KLAIM as follows:

$$
\begin{aligned}
& s_{rating} :: \langle \text{“}\mathit{ratingList}\text{”}, s_{party\_1}, m_1 \rangle \\
& \qquad \mid \langle \text{“}\mathit{rating}\text{”}, 1, s_{party\_1}, s_{rater}, v_{rating} \rangle \\
& \qquad \mid \ldots \mid \langle \text{“}\mathit{rating}\text{”}, m_1, s_{party\_1}, s'_{rater}, v'_{rating} \rangle \\
& \qquad \ldots \\
& \qquad \mid \langle \text{“}\mathit{ratingList}\text{”}, s_{party\_n}, m_n \rangle \\
& \qquad \mid \langle \text{“}\mathit{rating}\text{”}, 1, s_{party\_n}, s''_{rater}, v''_{rating} \rangle \\
& \qquad \mid \ldots \mid \langle \text{“}\mathit{rating}\text{”}, m_n, s_{party\_n}, s'''_{rater}, v'''_{rating} \rangle \\
& \| \\
& s_{party\_1} :: A_{party}(b_1) \mid C_{providerList\_1} \\
& \| \\
& \ldots \\
& \| \\
& s_{party\_n} :: A_{party}(b_n) \mid C_{providerList\_n}
\end{aligned}
$$

Each system party i is rendered as a KLAIM node whose locality name is s_{party_i} and, similarly, the rating server is rendered as a node, with locality name s_{rating}, as well. The rating server provides the list of ratings concerning each party in the system. Such list is rendered in KLAIM as a set of tuples of the form $\langle \text{“}\mathit{rating}\text{”}, i, s_{party\_j}, s_{rater}, v_{rating} \rangle$ denoting that s_{party_j} received the rating value v_{rating} from s_{rater}, such rating value is the i-th element of the list concerning s_{party_j}. A tuple of the form $\langle \text{“}\mathit{ratingList}\text{”}, s_{party\_j}, m_j \rangle$, denoting that the list for s_{party_j} has length m_j, is used to read the whole list. Such tuple is used also for coordinating read and write operations on the list: actions **in**, **out** and **read** act on this tuple by implementing a readers-writers lock mechanism. Finally, each party node s_{party_j} contains a process $A_{party}(b_h)$, specifying the operations carried out by the party, and a list $C_{providerList\_j}$ of providers known by the party.

The list $C_{providerList\_j}$ is rendered in KLAIM as follows:

$$C_{providerList\_j} \triangleq \langle \text{“}providerList\text{”}, n\rangle \mid \langle \text{“}provider\text{”}, 1, s_{party\_1}\rangle \mid \ldots \mid \langle \text{“}provider\text{”}, n, s_{party\_n}\rangle$$

The tuple $\langle \text{“}providerList\text{”}, n\rangle$ denotes that there are n providers in the list, whereas the tuples of the form $\langle \text{“}provider\text{”}, i, s_{party\_i}\rangle$ denote that s_{party_i} is the i-th provider in the list.

Depending on the processes running on parties node, each party s_{party_j} can play different roles: it can provide resources, require resources, or both. We consider here the more complete case, where $A_{party}(b)$ is defined as follows:

$$A_{party}(b) \triangleq A_{provider}(b) \mid A_{client}$$

The parameter b denotes the party's behaviour, i.e. the value b is the distribution parameter θ determining party's behaviour as discussed in Section 2.5. In the rest of the section we discuss the specification of the processes $A_{provider}(b)$ and A_{client}. We also explain which parts of the system specification have to be customized for the analysis of different reputation systems.

The process for the provider role is defined as follows:

$$\begin{aligned} &A_{provider}(b) \triangleq \\ &\quad \text{// wait for a new resource request} \\ &\quad \mathbf{in}(\text{“}request\text{”}, !l_{applicant})@\mathbf{self}; \\ &\quad \text{// provide the resource and restart} \\ &\quad (\, A_{provideResource}(b, l_{applicant}) \mid A_{provider}(b)\,) \end{aligned}$$

A provider waits for requests from clients and when a new request arrives it provides the resource to the client. The processing of a resource request is based on the definition of the process $A_{provideResource}$, which provides to the applicant a resource whose quality depends on provider's behaviour. The definition of such process may vary from a reputation system to another. In fact, this is a parameter that must be specified when considering a specific reputation system.

The process for the client role, instead, is defined as follows:

$A_{client} \triangleq$
  // initialise the tuple containing the most trusted party
  out(“$mostTrusted$”, **self**, NO_ONE)@**self**;
  // read the list of known providers
  read(“$providerList$”, !m)@**self**;
  for $j = 1$ **to** m {
    read(“$provider$”, j, !$l_{provider}$)@**self**;
    // compute the reputation of the j-th provider
    $A_{evaluateReputation}(l_{provider})$;
    // retrieve the computed reputation value
    in(“$reputation$”, !rep)@**self**;
    // update the most trusted party
    in(“$mostTrusted$”, !$l_{trusted}$, !$repMT$)@**self**;
    if ($repMT < rep$) **then**{
      out(“$mostTrusted$”, $l_{provider}, rep$)@**self**
    } **else** {
      out(“$mostTrusted$”, $l_{trusted}, repMT$)@**self**
    };
  };
  // get the most trusted provider and then check its reputation
  in(“$mostTrusted$”, !$l_{trusted}$, !$repMT$)@**self**;
  if (MIN_REPUTATION $\leqslant repMT$) **then**{
    // send the resource request to the provider
    out(“$request$”, **self**)@$l_{trusted}$;
    // receive the resource
    in(“$resource$”, !$quality$)@**self**;
    // check resource's quality and rate the provider
    $A_{rate}(b, quality, l_{trusted})$
  };
  $A_{client}(b)$

A client cyclically determines the most trustworthy provider and re-

quests a resource to it. To this aim, the client first initializes the tuple containing the most trustworthy provider using the constant NO_ONE that denotes that there exists no provider for a requested resource. Then, the client computes the reputation of each known provider through the process $A_{evaluateReputation}$, whose skeleton definition is as follows:

$A_{evaluateReputation}(l) \triangleq$
... possible variables initialisation ...
// read the rating values of the provider l
read("$ratingList$", l, $!m$)@s_{rating};
for $j = 1$ **to** m {
    // get an element of the list
    read("$rating$", j, l, $!l_{rater}$, $!rating$)@s_{rating};
    ... use $rating$ to compute the reputation of l ...
};
... compute reputation ...
out("$reputation$", rep)@**self**

The above process differs from a reputation system to another and is, indeed, a parameter that must be specified to consider a given reputation system. Once the client has selected the provider, it checks if the provider's reputation is above a given threshold. The constant MIN_REPUTATION is the system's parameter that specifies the minimum reputation required by the client for an interaction with a provider. If the provider's reputation score is above the MIN_REPUTATION value, the client sends a resource request to the provider and waits for the resource. Finally, once the client has received the resource, the process A_{rate} evaluates the quality of the obtained resource and rates the provider accordingly. As in the case of process $A_{evaluateReputation}$, the definition of the process A_{rate} may differ from a reputation system to another.

4.1.1 Beta and ML Reputation Systems Specification

For our analyses we consider the Beta and the ML models (already introduced in Section 2.5) in the case of binary ratings (i.e., an interaction is either 'satisfactory' or 'unsatisfactory'). We report here the KLAIM processes defining such reputation models. Since both reputation models rely on binary ratings and we use a common notion of quality (i.e., a resource is either 'good' or 'bad'), the following processes are the same for both models:

$$
\begin{array}{l}
A_{provideResource}(b, l_{applicant}) \triangleq \\
\quad random\_value := random(1); \\
\quad \textbf{if } (random\_value > b) \textbf{ then}\{ \\
\qquad \text{// provide a bad quality resource} \\
\qquad \textbf{out}(\text{“}resource\text{”}, \text{“}bad\text{”})@l_{applicant} \\
\quad \} \textbf{ else } \{ \\
\qquad \text{// provide a good quality resource} \\
\qquad \textbf{out}(\text{“}resource\text{”}, \text{“}good\text{”})@l_{applicant} \\
\quad \}
\end{array}
$$

$$
\begin{array}{l}
A_{rate}(quality, l) \triangleq \\
\quad \text{// get the number of ratings concerning } l \\
\quad \textbf{in}(\text{“}ratingList\text{”}, l, !m)@s_{rating}; \\
\quad \text{// check the quality of the resource obtained and send} \\
\quad \text{// the rating for the provider to the rating server} \\
\quad \textbf{if } (quality == \text{“}good\text{”}) \textbf{ then}\{ \\
\qquad \textbf{out}(\text{“}rating\text{”}, m+1, l, \textbf{self}, 1)@s_{rating} \\
\quad \} \textbf{ else } \{ \\
\qquad \textbf{out}(\text{“}rating\text{”}, m+1, l, \textbf{self}, 0)@s_{rating} \\
\quad \}; \\
\quad \textbf{out}(\text{“}ratingList\text{”}, l, m+1)@s_{rating}
\end{array}
$$

The process $A_{provideResource}(b, l_{applicant})$ implements the probabilistic be-

haviour of the providers, indeed resource quality is determined probabilistically using the distribution determined by the parameter b. The process $A_{rate}(quality, l)$, as already said, is used by clients to rate providers accordingly to resource quality. By the specification of such process it is possible to notice how clients interact (in writing mode) with a provider's rating list in mutual exclusion, since the $ratingList$ tuple acts as lock that must be acquired to add a new rating and released at the end of this operation.

The process for the computation of the reputation by means of the Beta model is defined as follows:

$A_{evaluateReputation}(l) \triangleq$
  $rep := 0;$
  $positive := 0;$
  // read the rating values of the provider l
  read("$ratingList$", l, !m)@**self**;
  if $(m > 0)$ **then**{
    for $j = 1$ **to** m {
      // get an element of the list
      read("$rating$", $j, l,$!l_{rater}, !$rating$)@**self**;
      // count the number of positive ratings
      if $(rating == 1)$ **then**{
        $positive := positive + 1;$
      }
    $rep := (positive + 1) / (m + 2);$
    }
  } **else** {
    $rep := NO\_RATINGS;$
  }
  // compute and return the reputation
  out("$reputation$", rep)@**self**

Notably, in case of a provider with no rating value assigned, a de-

fault value NO_RATINGS is used to determine the provider's reputation score. Such value is a system's parameter that must be specified and that can be tuned in the reputation system under verification.

Instead, the process for the ML model is defined as follows:

$$
\begin{aligned}
&A_{evaluateReputation}(l) \triangleq \\
&\quad rep := 0; \\
&\quad positive := 0; \\
&\quad \text{// read the rating values of the provider } l \\
&\quad \mathbf{read}(\text{“}ratingList\text{”}, l, !m)@\mathbf{self}; \\
&\quad \mathbf{if}\ (m > 0)\ \mathbf{then}\{ \\
&\qquad \mathbf{for}\ j = 1\ \mathbf{to}\ m\ \{ \\
&\qquad\quad \text{// get an element of the list} \\
&\qquad\quad \mathbf{read}(\text{“}rating\text{”}, j, l, !l_{rater}, !rating)@\mathbf{self}; \\
&\qquad\quad \text{// count the number of positive ratings} \\
&\qquad\quad \mathbf{if}\ (rating == 1)\ \mathbf{then}\{ \\
&\qquad\qquad positive := positive + 1; \\
&\qquad\quad \} \\
&\qquad rep := positive\,/\,m; \\
&\qquad \} \\
&\quad \}\ \mathbf{else}\ \{ \\
&\qquad rep := NO\_RATINGS; \\
&\quad \} \\
&\quad \text{// compute and return the reputation} \\
&\quad \mathbf{out}(\text{“}reputation\text{”}, rep)@\mathbf{self}
\end{aligned}
$$

4.2 Stochastic specification and analysis

In this section, we demonstrate how the KLAIM specification presented in the previous section can support the analysis of trust and reputation systems. Our approach relies on formal tools, such as stochastic simulation, modal logics and model checking, that permit expressing and evaluating performance measures in terms of logical formulae.

We enrich the KLAIM specification introduced in the previous section with stochastic aspects. As an excerpt of the STOKLAIM specification, we report below the stochastic definition of process $A_{evaluateReputation}$:

$$
\begin{array}{l}
A_{evaluateReputation}(l) \triangleq \\
\quad \ldots \\
\quad \mathbf{read}(\text{“}ratingList\text{”}, l, !m)@s_{rating} : \lambda_1 \; ; \\
\quad \mathbf{for}\ j = 1\ \mathbf{to}\ m\ \{ \\
\qquad \mathbf{read}(\text{“}rating\text{”}, j, l, !l_{rater}, !rating)@s_{rating} : \lambda_1 \; ; \\
\qquad \ldots \\
\quad \}; \\
\quad \ldots ; \\
\quad \mathbf{out}(\text{“}reputation\text{”}, rep)@\mathbf{self} : \lambda_2
\end{array}
$$

The actions highlighted by a gray background are those annotated with rates, where $\lambda_1 = 37.0$ and $\lambda_2 = 1400.0$. These rates assume that an ADSL connection (1.5 Mbit/s downstream and 0.5 Mbit/s upstream) is used and that the operation of reading a rating costs as the transfer of 5KB of data. For the local writing of a reputation value, we assume that the operation is executed on a local hard drive.

4.2.1 Simulations

The results of some simulation runs of the STOKLAIM specification, performed by using SAM, are reported in Figures 6, 7, 8 and 9 (see Appendix A for further results). The results are averaged across 1500 simulation runs[1]. The charts present the trend of the reputation of a given party in the system and the error committed by the reputation model in the computation of reputation scores. In the charts an horizontal line denotes the *true* paty's behaviour. The x-axis reports the numbers of rating values used to compute reputation scores. The y-axis reports both the reputation scores and the error committed. As measure of the error we

[1]On an Apple MacBook Pro computer (2.4 GHz Intel Core 2 Duo and 4 GB of memory) simulation of a single run needs an average time of 0.04 seconds.

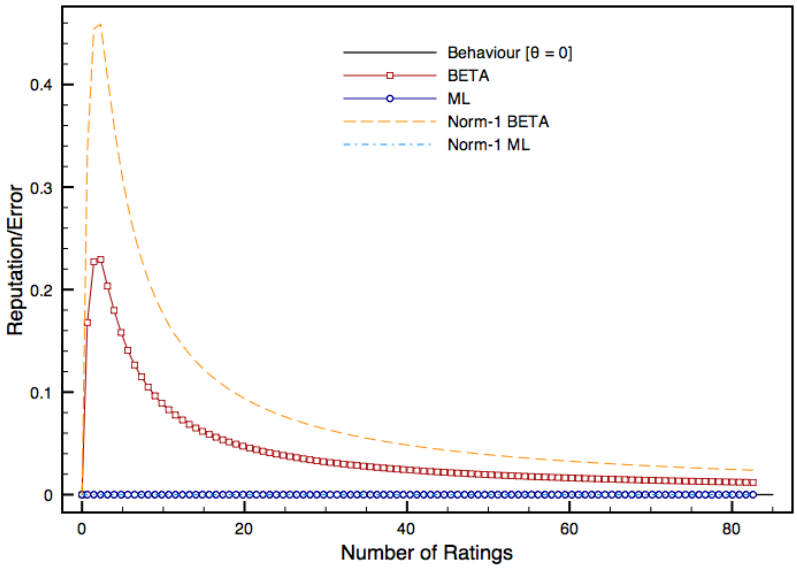


Figure 6: Reputation and error trends for parties with behaviour $\theta = 0$ and no initial ratings assigned

used the norm-1 distance between probability distributions (we already used it in Chapter 3 as a possible definition for loss functions). In our analysis we compare the Beta model and the ML model in case of binary ratings by studying:

- the trend of the reputation score computed by the models, reported in the charts as 'BETA' and 'ML' lines, respectively;
- the trend of the error committed by the models, reported in the charts as 'Norm-1 BETA' and 'Norm-1 ML' lines, respectively.

We compare the two models both in presence and in absence of initials ratings. Such initials ratings are used to determine the initials reputation scores for parties, before any interaction in the system.

From simulation data we observe that the ML model performs better than the Beta model if we consider the computed reputation score. In-

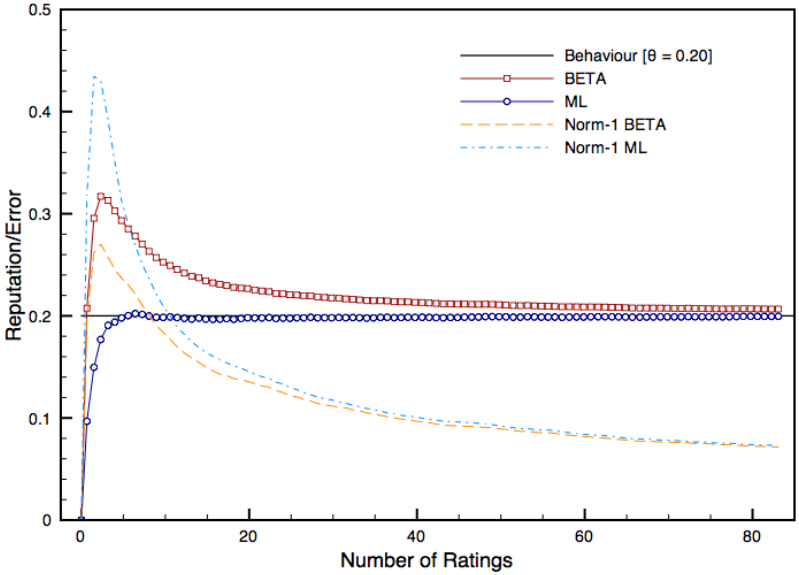


Figure 7: Reputation and error trends for parties with behaviour $\theta = 0.20$ and no initial ratings assigned

deed the reputation score computed by the ML model is always closer to the true party's behaviour than the reputation score computed by the Beta model. Such evaluation does not give us a complete information about models performances. Indeed looking at the error committed by the models, we observe that the ML model does not perform always better than the Beta model. Roughly speaking, for behaviour values lower than $\theta = 0.9$ and higher than $\theta = 0.1$ the error committed by the Beta model is always lower than the one committed by the ML model. It happens the opposite for behaviour values lower than $\theta = 0.1$ and higher than $\theta = 0.9$. Finally, we observe that the Beta model tend to over- or under- estimate party's behaviour. Specifically, the Beta model underestimates behaviours higher than $\theta = 0.5$, and overestimates behaviours lower than $\theta = 0.5$. We believe that this is an unwanted behaviour for trust and reputation models. Indeed the Beta model gives an advantage

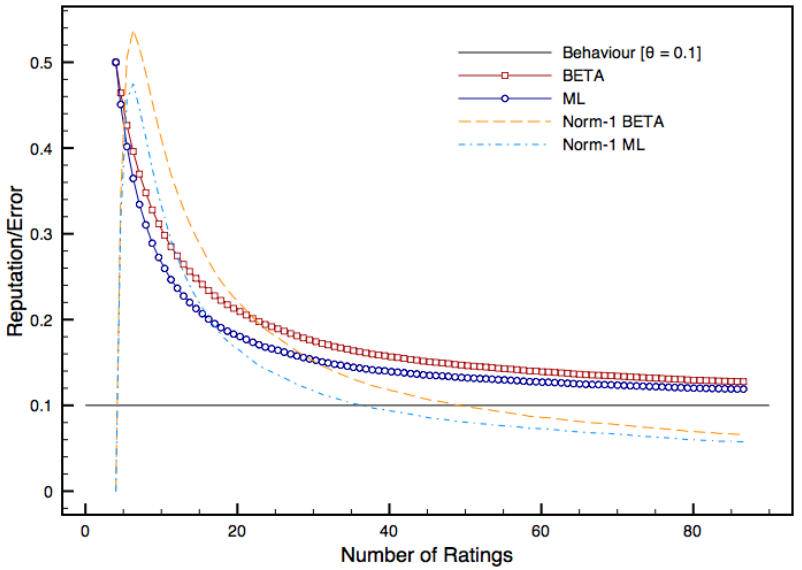


Figure 8: Reputation and error trends for parties with behaviour $\theta = 0.10$ and 4 initial ratings fixing initial reputation score to 0.5

to parties with a bad behavior, computing for them a reputation score (in average) higher than their true behaviour.

Figures 8 and 9 report some simulation results in the case of initial rating values. The charts report analysis results in presence of four initial rating values for each party that fix parties' initial reputation to 0.5. We observe that also in this case the ML model performs better than the Beta model in computing reputation scores. Moreover, we notice that the ML model benefits from initial ratings if compered with the Beta model. Specifically, we observe that for behaviour values lower than $\theta = 0.3$ and higher than $\theta = 0.7$ the error committed by the ML model is always lower than the one committed by the Beta model. The Beta model benefits from initial ratings only for behaviour values lower than $\theta = 0.6$ and higher than $\theta = 0.4$. That happens because the true behaviour of the party is close to the initial reputation score, i.e. party's behaviour is close

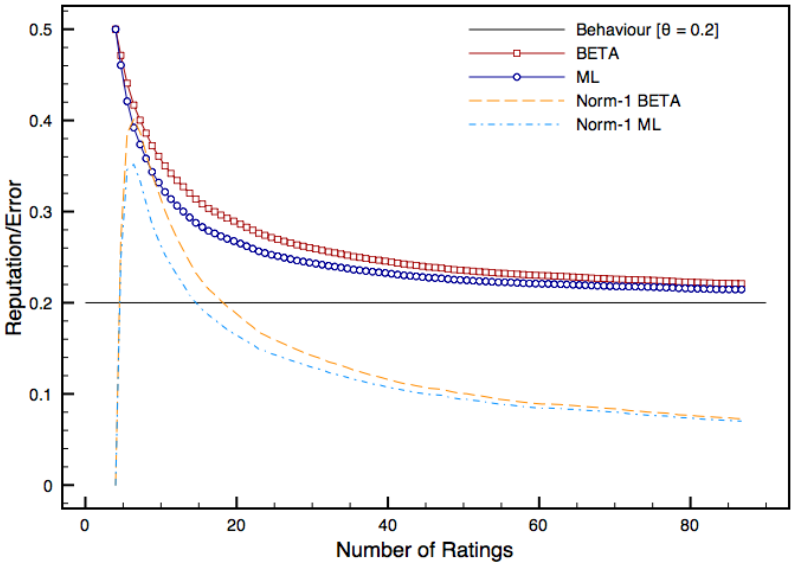


Figure 9: Reputation and error trends for parties with behaviour $\theta = 0.20$ and 4 initial ratings fixing initial reputation score to 0.5

to $\theta = 0.5$. Indeed, the Beta model has a bias towards reputation values close to 0.5.

To sum up the results of our analysis allow us to argue that initial ratings can improve the models' performances, but the number and the value of such ratings has to be choosen carefully in order to get the desired result. Notice that initial ratings play a different role for different behaviours, i.e. behaviours close[2] to the initial reputation value are inferred with a lower error than in the case of no initial ratings. It happens the opposite for others behaviours, for that the error committed by the models is higher compared to a system with no initial ratings. Initial ratings could be used as an incentive for good behaviour, i.e. fixing a low initial reputation score should push parties to behave well if they want

[2]The term *close* denoting the distance between probability distributions may not have a clear meaning. Anyway, in this case, where Bernoulli distributions are used and a single parameter identifies a distribution, we believe that this term makes sense.

"the reputation of s_{party_i} converges to its actual behaviour θ_i within time t"		
Initials Ratings	Beta Model	ML Model
0	1	1
2	0.804444828548	1
4	0.00745483834375	0.802581118962

Table 2: Satisfaction probability for party's behaviour $\theta = 1$ and time $t = 50$

to achieve an high reputation score.

4.2.2 Model Checking

We analyse some properties of reputation systems by formalising them as MOSL formulae that we have verified over the STOKLAIM specification by means of the SAM tool. The first property we analyse is "*the reputation of s_{party_i} is currently within $[\theta_i + \delta, \theta_i - \delta]$*", with θ_i denoting the behaviour of s_{party_i} and δ denoting the error interval. It is expressed in MOSL by the following formula ϕ_{conv}:

$$\begin{aligned}\phi_{conv} = \langle \text{“}reputation\text{”}, s_{party\_i}, !rep\rangle @ s_{rating} \\ \rightarrow (\theta_i + \delta \geq rep \ \ \wedge \ \ rep \geq \theta_i - \delta)\end{aligned}$$

This formula relies on the *consumption* operator $\langle T \rangle @ s \rightarrow \phi$, which is satisfied whenever a tuple matching template T is located at s and the remaining part of the system satisfies ϕ. Hence, formula ϕ_{conv} is satisfied if and only if a tuple $\langle \text{“}reputation\text{”}, s_{party\_i}, v_{rep} \rangle$ is stored in the node s_{rating} and the reputation value v_{rep} is equal to the party's behaviour θ_i up to a given error δ. Notice that the KLAIM model of the reputation system has been slightly modified to enable this analysis. In particular, the client process updates tuples of the form $\langle \text{“}reputation\text{”}, s_{party\_i}, v_{rep} \rangle$ in the rating server every time it computes a provider's reputation score. Thus, exploiting the previous formula, we check the property "*the reputation of s_{party_i} converges to its actual behaviour within time t*" that is defined as $true\ U^{\leq t} \phi_{conv}$, where the *until* formula $\phi_1 U^{\leq t} \phi_2$ is satisfied by all the runs that reach within t time units a state satisfying ϕ_2 while only traversing states that satisfy ϕ_1. The model checking analysis has been

"the reputation of s_{party_i} converges to its actual behaviour θ_i within time t"		
Initials Ratings	Beta Model	ML Model
0	0.696688528852	0.651790070646
2	0.404763836437	0.692622253392
4	0.200094638272	0.402222414274

Table 3: Satisfaction probability for party's behaviour $\theta = 0.9$ and time $t = 50$

then performed by estimating the total probability of the set of runs satisfying such formula, the maximal time t has been set to 50 seconds and the error δ to 0.05. The average amount of ratings available for the computation after 50 seconds is 45. The parameters ϵ and p of the model checking algorithm used by SAM (see Section 2.4.2) have been both set to 0.05.

In Tables 2 and 3 some analysis results are reported. In Table 2 we consider a party with behaviour $\theta = 1$ and the presence of 0, 2 or 4 initial ratings that fix parties' initial reputation to 0.5 (as in the previous analysis). We observe that the ML model always performs better than the Beta model, as expected. In particular for 4 initial ratings the Beta model is (almost) never able to compute a reputation score in the fixed interval. Instead the ML model is still able to compute a good approximation of party's behaviour. In Table 3 we consider party with behaviour $\theta = 0.9$ and the presence of 0, 2 or 4 initial ratings in the system. In this case we know from the simulation results that the ML model performs better than Beta model both in computing reputation scores and for the error committed in the estimation. However, we observe that in presence of no ratings value at the start of the system, the Beta model achieves better results. Indeed, the satisfaction probabilty of the formula for the Beta is higher than that for the ML model. This means that, even though the error committed by the ML model is lower in average, the Beta model is slightly more accurate in the estimation. We also notice that the ML model achieves better results in presence of 2 initial ratings than with no initial ratings. In this case the probability is almost the same that for the

"the reputation of s_{party_i} goes below a given threshold within time t"		
Initials Ratings	Beta Model	ML Model
0	0.999117766207	0.999626050639
2	0.995051490746	0.999287194351
4	0.993018353016	0.996068059612
6	0.985224658384	0.991493499719
8	0.977092107464	0.986241227249

Table 4: Satisfaction probability for party's behaviour $\theta = 0.1$, threshold 0.35 and time $t = 20$

"the reputation of s_{party_i} goes below a given threshold within time t"		
Initials Ratings	Beta Model	ML Model
0	0.948967035531	0.967265275102
2	0.885939765897	0.95015303254
4	0.822065355543	0.883567771879
6	0.742095271491	0.826470487291
8	0.66941059764	0.742942412212

Table 5: Satisfaction probability for party's behaviour $\theta = 0.25$, threshold 0.35 and time $t = 20$

Beta model without initial ratings. Concluding, the results of the model checking confirm the simulation outcomes, pointing out that the ML model is less stable than the Beta model when few ratings are available.

The second property we checked is "*the reputation of s_{party_i} goes below a given threshold within time t*" that is formalised in MoSL as

$$true\ U^{\leq t}\ \big(\ \langle \text{“}reputation\text{”}, s_{party\_i}, !rep\rangle @ s_{rating} \rightarrow threshold \geq rep\ \big)$$

The threshold may represent, e.g., the minimum reputation that a provider must have in order to be considered trustworthy for an interaction or the minimum reputation required to interact in the system. In Tables 4 and 5 some analysis results are reported . For this analysis we set t to 20 seconds, the average amount of ratings available for the computation after 20 seconds is 21. Also in this case, the parameters ϵ and p of the model checking algorithm used by SAM have been both set to

0.05. In Table 4 we consider a party with behaviour $\theta = 0.1$, a threshold value of 0.35 and the presence of 0, 2, 4, 6 or 8 initial ratings in the system. We observe that the satisfaction probabilities are very close for the two models, but the ML model always achieves better results than the Beta model. In Table 5 we consider a party with behaviour $\theta = 0.25$, a threshold value of 0.35 and the presence of 0, 2, 4, 6 or 8 initial ratings in the system. In this case we observe that the ML model performs better than the Beta model and the satisfaction probabilities are not so close, between the two models, as in the previous case. We noticed, from simulation results, that the Beta model performs better for behaviours higher then $\theta = 0.1$ for what concern error estimation, but from this analysis we obtain a further information. Even though the error committed by the ML model is higher, such model is faster than the Beta model in convergence. Thus it achieves better results in this analysis and it is preferable when bad behaivours must be detected.

We conclude the section, by noticing that the combined use of model checking and simulation techniques permits to analyse systems' details that a single approach would not be able to detect.

Chapter 5

A Network-aware Evaluation Environment for Reputation Systems

Once a reputation system has to be deployed in a network environment, several details has to be taken into account. This calls for an engineering approach for describing, implementing and evaluating reputation systems while taking into account real-word implementation details of such systems and of the network environment where they have to be deployed.

In this chapter, we address this issue by proposing a software tool for network-aware evaluation of reputation systems. More specifically, on the one hand, we provide a framework for rapidly developing Java-based implementations of reputation system models and for easily configuring different networked execution environments on top of which the systems will run. On the other hand, we offer a software tool that automatically performs experiments on the reputation system implementations according to user-specified parameters; this enables the study of their behaviour while executing on given network infrastructures. The main novelty of our approach, with respect to other proposals in the literature with a similar aim, is that we allow the evaluation of im-

plemented reputation systems through experiments on real networks, rather than performing simulation of models of reputation systems that abstract from many details. In this way, given a specific network environment, we can study the system behaviour to find the configuration that better meets the system requirements by tuning its parameters (reputation model, response timeouts, resource quality evaluation, ratings aging, etc.). Moreover, since we consider reputation systems at implementation level, the analysed systems could be then directly used in the corresponding end-user applications.

The rest of the chapter is organized as follows. Section 5.1 describes the general overlay network for reputation systems set up by our tool. Section 5.2 describes the architecture and functional principles of our tool. Section 5.3 provides a brief overview of the tool component dealing with networking aspects. Section 5.4 presents the implementation aspects of the reputation models currently considered in this work. Finally, Section 5.5 reports on the analysis of a few reputation systems.

5.1 A general infrastructure for reputation systems

We consider in this chapter a centralised architecture graphically depicted in Figure 10, where parties can interact and exchange ratings. Such architecture is widely used for networked trust infrastructures and it is slightly different from the one introduced in Chapter 4. In this general infrastructure, a *rating server* collects ratings from system's parties and makes them publicly available, while a *search server* allows parties to find resource providers in the system. Every party can play the role of a client, of a provider, or both, and may offer different kinds of resources (services, computational and storage resources, etc.). Whenever a party needs a resource, first it queries the search server to get the list of parties providing it, and then retrieves from the rating server the ratings of each provider in the list. As usual, to choose a provider, it computes the reputation scores of each of them and selects the one with the highest reputation score. Finally, after the interaction, it rates the provider

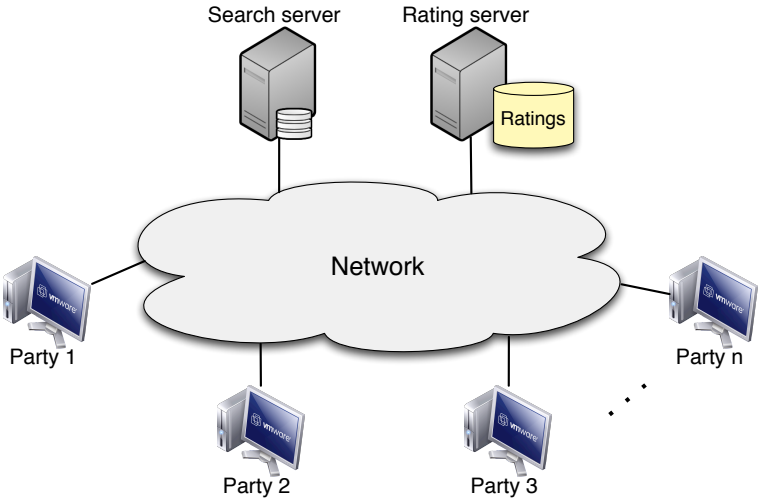


Figure 10: General infrastructure of a reputation system

according to the quality of the provided resource.

On top of the general infrastructure just described, different kinds of reputation system can be layered, which mainly differ for the model they use to aggregate ratings when computing reputation scores.

5.2 The NEVER tool

In this section, we present the architecture and the workflow of NEVER (Network-aware EValuation Environment for Reputation systems), graphically depicted in Figure 11. The NEVER tool consists of three main components: (1) the experiment manager, (2) the network infrastructuring (3) the reputation models library.

The *experiment manager* is the component playing the main role, because it is in charge of managing the execution of each experiment. An experiment consists of a user-specified number of runs, each run performed with the same configuration. The number of runs and their du-

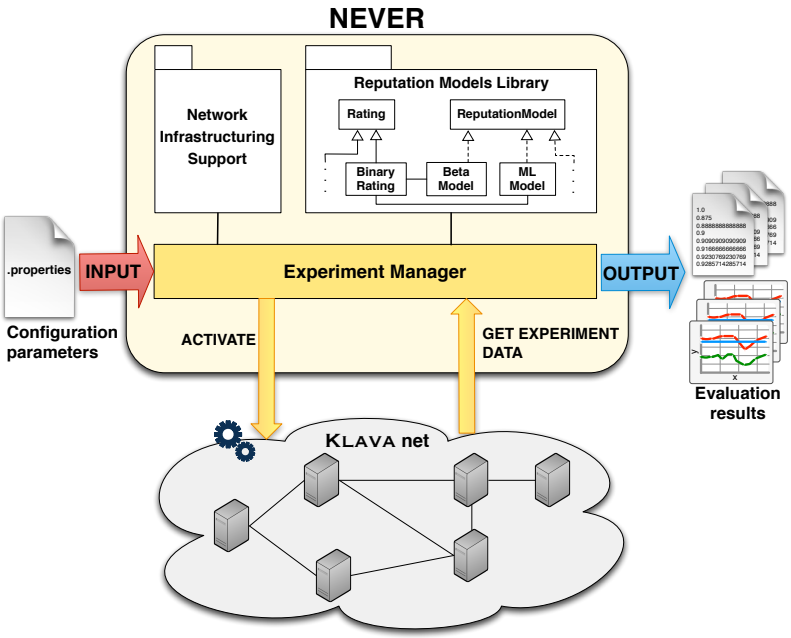


Figure 11: NEVER architecture and workflow

ration, together with other experiments characteristics, are defined by users through configuration parameters.

The *network infrastructuring support* provides the libraries (i.e., classes and interfaces) required to create and set up a KLAVA *net* (see Section 2.4.3 for further details on KLAVA) implementing the general infrastructure graphically depicted in Figure 10. Each element of the infrastructure is a node hosted by a (possibly remote and/or virtual) machine. The NEVER tool takes as input the addresses of the hosting machines and automatically activates the nodes setting up the wanted network infrastructure. We refer to Section 5.3 for further details on the network infrastructure library supporting our experiments.

The *reputation models library* acts as a framework allowing the user

to define the trust and reputation models under evaluation. The library is a Java package containing a number of abstract classes and interfaces necessary to implement the models. In this way, the NEVER tool is customizable and extendible by the user. More specifically, a reputation model is defined by a class implementing the ReputationModel interface and, possibly, a class extending the abstract class Rating. The former class defines how reputation scores are computed, which rating values are used by the system and how parties in the system evaluate the interactions. The latter class defines the kind of rating values and how to manage them. Thus, the addition of new reputation models to NEVER can be achieved by implementing ReputationModel and, if necessary, by extending Rating. We refer to Section 5.4 for further details on the reputation models library and on the models already available in NEVER.

We describe now the NEVER workflow, by lingering on the main features of the experiment manager component. The tool takes as input a set of *configuration parameters*, written in a *.properties* file as pairs of the form $key = value$. Such parameters are used by the experiment manager to instantiate and carry out an experiment. First, the manager creates the network on top of which will be run the experiment. Afterwards, a node is created for each of the two servers and for each party in the system. Once the network is set up, the reputation system (configured according to user's parameters) is deployed on the network and the experiment starts, i.e. network components are enabled and system parties interact and rate each other. During the activity of the network, data about interactions are stored in appropriate files for a later analysis. Experiment runs are repeated in order to reach the desired precision; thus, the manager starts and stops runs till the last run is accomplished. Afterwards, data are analysed and provided as output, both in form of textual files and charts[1]. We refer to Section 5.5 for a discussion about different experiments carried out with our tool.

We conclude this section by commenting on the relevant configuration parameters. Through such parameters it is possible to specify the

[1]The tool automatically generates charts by exploiting the Java library JFreeChart (freely available at `http://www.jfree.org/jfreechart/`).

number of parties in the system and the addresses of the machines where parties have to run. For each party, a new KLAVA node is automatically created and deployed in the associated hosting machine. The tool also supports a 'local only' modality, where all KLAVA nodes are deployed in the same machine running the tool. Such modality can be useful to compare reputation systems in presence or absence of networking aspects affecting the evaluation.

A specific configuration parameter is used to set the main reputation model, which is used during the experiment to drive the interactions among parties. In fact, when a party is looking for a provider of a specific resource, it computes the providers' reputations and selects for the interaction the most trusted one, i.e. the party (or one of the parties) with the highest reputation value. Besides the main model, it is possible to give a list of trust and reputation models to be compared during the experiment: each party's reputation is computed according to all models specified in such list. Values of party's reputation are returned for each run and, at the end of the experiment, as a mean value over all runs. Moreover, the user can require to randomly select the providers, by thus ignoring the choice of the providers based on the main reputation model. Such modality is indeed often used in our experiments, because it gives the opportunity of evaluating models performances by comparing party reputations on the basis of approximately the same amount of ratings for each party.

A group of configuration parameters regulates the parties' behaviour. The user specifies a set of possible party behaviours and the percentage of parties with each given behaviour. By means of such information, the experiment manager assigns a behaviour to each party. Moreover, it is possible to set the initial reputation of parties by specifying the values and the number of their initial ratings. Such ratings determine the initial parties reputation computed by the system. In the default case, parties' behaviour are assumed to be fixed, but a changeable behaviour can be configured. In such a case, the user sets when the variation has to happen and the magnitude of the variation. Currently, the variation implemented is negative, i.e. party's behaviour gets worse after the vari-

ation. The idea is that once a party achieves a high reputation score, it could then exploit such value for behaving badly in the next interactions. Several studies (see, e.g., [JI02; SS01; ZM00]) use similar approaches for the evaluation of reputation models.

Finally, the configuration parameters allow the user to set two threshold values: the maximum delay and the maximum waiting time. The first parameter sets the maximum delay after which a resource is considered unsatisfactory, i.e. once the party receives the resource it checks if the arrival time exceeds the maximum delay and, in such a case, a negative rating is given to the provider no matter the quality of the resource. The second parameter sets the maximum time that a party will wait for a resource; expired this time a new provider is selected by the party and no rating value is given. In this way, a party will not wait indefinitely for a resource.

The NEVER tool is developed in Java, by exploiting freely available third-party libraries. Source and binary files of NEVER can be found at `http://sysma.lab.imtlucca.it/tools/never/`.

5.3 Network infrastructuring support

The network infrastructuring support of NEVER provides an API that allows the experiment manager to create different networks underlying the reputation systems to be evaluated. To this aim, this tool element exploits the KLAVA library (see Section 2.4.3) and is implemented as a Java package. In this section, we present the functionalities of each component of the package and we describe its implementation in KLAVA. We illustrate how network components interact by showing pieces of code.

The network infrastructuring package specifies three different kinds of nodes that take part in the KLAVA net: a *rating server* node, a *search server* node and a *user* node. Each of these nodes implements a component of the infrastructure graphically depicted in Figure 10.

The rating server node serves as public database for collecting parties' ratings and executes the process RatingServerProcess. This process is in charge of collecting data produced by each experiment run.

The search server node assists parties while seeking a resource provider and executes the process SearchServerProcess (Listing 5.1). Such process waits for search requests sent by parties (line 5). Specifically, parties send requests, i.e. tuples tagged by the search_request string, stating the type of the resource they want from the provider. Afterwords, the SearchServerProcess looks in the local tuple space for available providers offering such resource (lines 11 and 21): for each provider matching the request, the process sends its address to the requesting party (line 17). The set of tuples sent by the SearchServerProcess to the party forms a list of provider addresses. When all providers have been checked[2], the SearchServerProcess closes the list by sending its length to the requesting party (line 25).

Listing 5.1: SearchServerProcess

```
1  // Wait for a new search request
2  KInteger n_providers = new KInteger( 0 ) ;
3  Locality loc_requester = new PhysicalLocality();
4  KString res_type = new KString() ;
5  in( new Tuple(new KString("search_request"),loc_requester,res_type), self);
6
7  // Scan the list of parties providing resources of type 'res_type'
8  Locality loc_provider = new PhysicalLocality();
9  Tuple templRead_nb = new Tuple(res_type, loc_provider);
10 templRead_nb.setHandleRetrieved(true);
11 KBoolean forallExpressionArgument =
12     new KBoolean( read_nb(templRead_nb, self ) ) ;
13 while ( forallExpressionArgument.booleanValue() ) {
14     // Increase the counter of providers
15     n_providers = new KInteger( n_providers.intValue() + 1 );
16     // Send the provider's address
17     out(new Tuple(new KString( "list" ),n_providers,loc_provider),
18         loc_requester);
19     templRead_nb.resetOriginalTemplate();
20     forallExpressionArgument =
21         new KBoolean( read_nb(templRead_nb, self) );
22 }
```

[2]The method setHandleRetrieved() allows the tuple templRead_nb to store all the tuples that it has matched (line 10), while method resetOriginalTemplate() is used to reinitialize to empty values the formal fields of templRead_nb (line 17) in order to use this template to retrieve another tuple in the next read_nb action (line 18).

```
23
24  // Send the length of the list of providers
25  out(new Tuple(new KString( "list_length" ), n_providers), loc_requester) ;
```

The user node implements a generic party; nodes of this kind interact to ask and provide resources and, after any interaction, rate each other. Two processes run on the user node[3]: the ProviderProcess (Listing 5.2) and the ClientProcess (Listing 5.3). The former process implements the functionalities of a provider: when a new resource request coming from a client is received (line 4), the resource is selected (line 7) and sent to the client (line 10). The resource selection consists of determining its quality according to the provider's behaviour; in fact, the actual provision of the resource is not relevant for our studies.

Listing 5.2: ProviderProcess

```
1   // Wait for a resource request
2   Locality requesterLoc = new PhysicalLocality();
3   KString res_type = new KString();
4   in(new Tuple(new KString("resource_request"), requesterLoc, res_type), self);
5
6   // Get resource quality according to provider's behaviour
7   KDouble res_quality = model.getResourceQuality(new_behaviour,rand);
8
9   // Send the resource
10  out(new Tuple(new KString("resource"), res_type, res_quality), requesterLoc);
```

The ClientProcess seeks providers for the resource it is looking for, and selects the most trusted one for the next interaction. It first sets the variable most_trusted_user to NO_ONE (lines 2-3) denoting that no provider has been selected. Then, it determines the resource type it wants to request (lines 6-7). The ClientProcess asks the search server to find a provider for a given resource type (line 10) and selects, among the providers returned by the search server, the most trusted one, i.e. the provider (or one of the providers) with the highest reputation score (line 16). Then, it checks if the reputation of such provider is higher than the minimum reputation value defined in the configuration file (line 20). If this check

[3]Depending on the processes running in its node, a party can play the role of a client, of a provider, or both. We consider here the latter case, which is the most general.

is positive, the process sends a request for the resource to the selected provider (lines 23-24), otherwise it starts again the procedure from the beginning. The waiting time of a requested resource is bounded by a time-out, request_time_out, specified in the configuration file (lines 30-31). When the resource is received the process computes a rating value for the provider (line 35) and sends it to the rating server (lines 41-44). The rating value is given on the basis of the resource quality and response time. The parameter max_waiting_time specified in the configuration file sets the maximum delay after which a resource is considered unsatisfactory.

Listing 5.3: ClientProcess

```
1  // Initialize the "most trusted user" tuple
2  out(new Tuple(new KString("most_trusted_user"), getPhysical(self),
3      new KDouble(NO_ONE), new KInteger(0)), self);
4
5  // Resource type is randomly selected
6  int res_num = (int) ((rand.Fran2() * (number_of_resource_types-1))+1);
7  KString res_type = new KString("type_"+res_num);
8
9  // Send the request to the search server and determine the most trusted user
10 searchProvider(res_type);
11
12 // Get the data of the most trusted user
13 Locality trusted_loc = new PhysicalLocality();
14 KDouble reputation_most_trusted = new KDouble();
15 KInteger number_of_ratings = new KInteger();
16 in(new Tuple(new KString("most_trusted_user"), trusted_loc,
17     reputation_most_trusted, number_of_ratings), self);
18
19 // Check if provider's reputation is higher than the minimal reputation
20 if (min_reputation <= reputation_most_trusted.doubleValue()) {
21
22     // Send a resource request to the provider
23     out(new Tuple(new KString("resource_request"), getPhysical(self),
24         res_type), trusted_loc);
25
26     // Wait the resource
27     long time_of_request=System.currentTimeMillis();
28     KDouble quality = new KDouble();
29     Rating provider_rating = model.createNonInitializedRate();
30     if ( in_t(new Tuple(new KString("resource"), res_type, quality),
```

```
31              self, request_time_out)) {
32          long time_of_service = System.currentTimeMillis();
33          long response_time = time_of_service−time_of_request;
34          // Check the quality of the obtained resource and rate the provider
35          try { provider_rating = model.rateProvider(quality,response_time); }
36          catch (MalformedRateValueException e) {
37              e.printStackTrace();
38              System.exit(1);
39          }
40          long ratingTime = System.currentTimeMillis();
41          out(new Tuple(getPhysical(self), trusted_loc, provider_rating.getValue(),
42              new KString(Long.toString(ratingTime)) ),
43          UserNode.rating_serverLogLoc);
44      }
45  }
```

5.4 Trust and reputation system models implementation

In this section, we provide some details about trust and reputation models already implemented in NEVER; this would also serve as a guide for using the framework to implement new models to be evaluated.

As we have shortly discussed in Section 5.2, it is possible to implement trust and reputation models in NEVER (as the ones introduced in Section 2.5) through the reputation models library. The first step is to create a class implementing the ReputationModel interface, whose main methods are the following:

- **public void** setWindow(**int** size);
- **public void** setForgettingFactor(**double** value);
- **public** KDouble evaluateReputation(Vector<Rating> ratings, Vector<Locality> raters);
- **public** KDouble getResourceQuality(**int** behaviour, Ran2 rand);

The method setWindow takes as input an integer that fixies the number of ratings, among the available ones, to use for the computation of party's

reputation score. The most recent ratings are used for the computation. The method setForgettingFactor set the value of the parameter called forgetting factor (see Section 2.5), it take as input a value in the interval $[0, 1]$. The method evaluateReputation is the basis of any reputation model. It takes as input a list of rating values (Vector<Rating> ratings) and the corresponding list of raters (Vector<Locality> raters), and returns as output the reputation score of the ratee. Ratings in the vector are sorted from the newest to the oldest; such organization is useful for models that discriminate ratings depending on their age. Finally, method getResourceQuality is used to determine the quality of the resource to be provided to the client. This value corresponds to the outcome of the interaction, and depends on the model implemented and on the set of rating values in use.

Now, we show how the Beta and ML models outlined in Section 2.5 are implemented in NEVER, by focussing on the code of the method evaluateReputation that specifies how rating values are used to compute party's reputation in the models. We start from the implementation of the ML model (Listing 5.4). The method evaluateReputation first checks the number of available ratings (line 3): if there are no ratings the computation does not take place and the default reputation value is used by the system, i.e. an empty KDouble() object is returned and the system uses as party's reputation the value set in the configuration file (parameter no_rating_reputation). A second check (line 6) is done on the window's size: if the number of rating values is bigger than the window's size, only the newer ratings are used. The last part of the code (lines 9-15) computes the party's reputation. In case of binary ratings, in the ML model this means to simply divide the number of satisfactory interactions by the total number of interactions (see equation (2.15) in Section 2.5). Finally, the computed reputation value is returned as a result by the method (line 16).

Listing 5.4: evaluateReputation (MLModel.java)

```
1  double num_of_ratings = ratings.size();
2  double num_positive_ratings = 0;
3  if ( num_of_ratings == 0){
4       return new KDouble();
5  }
```

```
6  if ( (WINDOW != 0) && (num_of_ratings > WINDOW)){
7      num_of_ratings = WINDOW;
8  }
9  for (int i = 0; i < num_of_ratings; i++) {
10     KInteger rating_value = (KInteger) ratings.get(i).getValue();
11     if ( rating_value.intValue() == POSITIVE_RATE_VALUE ){
12         num_positive_ratings++;
13     }
14 }
15 double ml_reputation = num_positive_ratings/num_of_ratings ;
16 return new KDouble(ml_reputation);
```

We now examine the code implementing the Beta model (Listing 5.5). In the Beta model, party's reputation is computed as the expected value of Beta distribution (see equation (2.11) in Section 2.5). We show only the last two lines of the method evaluateReputation, since the first part of the code, where it is checked the number of available ratings and the window's size, is common among all considered models.

Listing 5.5: evaluateReputation (BetaModel.java)

```
1  double beta_reputation = (num_positive_ratings+1)/(num_of_ratings+2) ;
2  return new KDouble(beta_reputation);
```

The last kind of models we implemented makes use of the forgetting factor. The code shown in Listing 5.6 is the final part of the method evaluateReputation implemented by the class BetaModelForgetting; the implementation of the same method within class MLModelForgetting is similar. Each rating value here is weighted according to its age (lines 4 and 6). The weight of each rating is given by the value λ^i, where λ (i.e. LAMBDA, in the code) is the forgetting factor and i denotes rating's age.

Listing 5.6: evaluateReputation (BetaModelForgetting.java)

```
1  for (int i = 0; i < num_of_ratings; i++) {
2      KInteger rating_value = (KInteger) ratings.get(i).getValue();
3      if ( rating_value.intValue() == POSITIVE_RATE_VALUE ){
4          num_positive_ratings = num_positive_ratings + Math.pow(LAMBDA, (i));
5      } else {
6          num_negative_ratings = num_negative_ratings + Math.pow(LAMBDA, (i));
7      }
8  }
9  double beta_rep =
```

```
10      (num_positive_ratings+1)/(num_negative_ratings+num_positive_ratings+2) ;
11  return new KDouble(beta_rep);
```

To implement a reputation model, it is also needed to provide the implementation of rating values. The models currently available in NEVER use *binary ratings*, i.e. each interaction can be rated either 'unsatisfactory' or 'satisfactory'. In order to define another kind of rating, a new class must be created as a subclass of the abstract class Rating (Listing 5.7).

Listing 5.7: Rating.java

```
1  public abstract class Rating implements Comparable<Rating>{
2      protected TupleItem value;
3      protected long timestamp;
4
5      public abstract void setValue (TupleItem value)
6              throws MalformedRateValueException;
7
8      public TupleItem getValue(){
9          return this.value;
10     }
11     public void setTime(KString time){
12         this.timestamp = Long.parseLong(time.toString());
13     }
14     public long getTime(){
15         return timestamp;
16     }
17     ...
18 }
```

This class defines four methods. Methods setTime and getTime are used to set and retrieve ratings' timestamps. Such values are used for sorting the vector containing rating values. Methods getValue and setValue return and set the value of the rating respectively. In particular, a new rating class has to implement the method setValue that is declared abstract; its implementation should check the format of the rating value.

5.5 NEVER at work

In this section we show which data NEVER returns as output and in which formats the output is provided. For illustration purpose, we run

an experiment with the following parameters: ten[4] parties are active in the system, data are averaged over fifty runs, each run lasts thirty five minutes. Five possible behaviours are defined in the system, all modelled as Bernoulli distributions of parameter θ. The possibles values of θ are: $0.05, 0.3, 0.5, 0.7, 0.95$. The twenty percent of the parties assumes one of the possible behaviours, that is for each possible θ there are two parties with such behaviour. Parties' behaviours are set to change after fifteen interactions, specifically each party updates its behaviour to $\theta = \theta - 0.4$, that is parties assume a worst behaviour than the initial one. If parties' initial behaviour θ is less than 0.4 the new behaviour is set to $\theta = 0$.

No initial reputation is set for parties: when a party is found to have no ratings is assumed to have a reputation score of 0.5. We recall to the reader that the parameter no_rating_reputations is used to set such value. Moreover, no minimum reputation values is set for interacting with a party (parameter min_reputation), that is to interact with a party no threshold is fixed about its reputation score.

The main reputation model is defined to be a ML model without forgetting factor and window. Instead the list of models to compare contains six models:

- ML model without window and forgetting factor;
- ML model with window=25 and without forgetting factor;
- ML model without window and with forgetting factor=0.9;
- Beta model without window and forgetting factor;
- Beta model with window=25 and without forgetting factor;
- Beta model without window and with forgetting factor=0.9.

At the end of the experiment, NEVER returns as output both a graphical representation of data and a textual list of data. Textual output is provided in order to permit data manipulation without re-executing experiments, so that different graphical representations can be used for data

[4]Given a limited number of physical machines at our disposal, to perform experiments with this number of parties, we have deployed KLAVA nodes on virtual machines running on the cloud IaaS platform Zimory Enterprise Cloud [Gmb12].

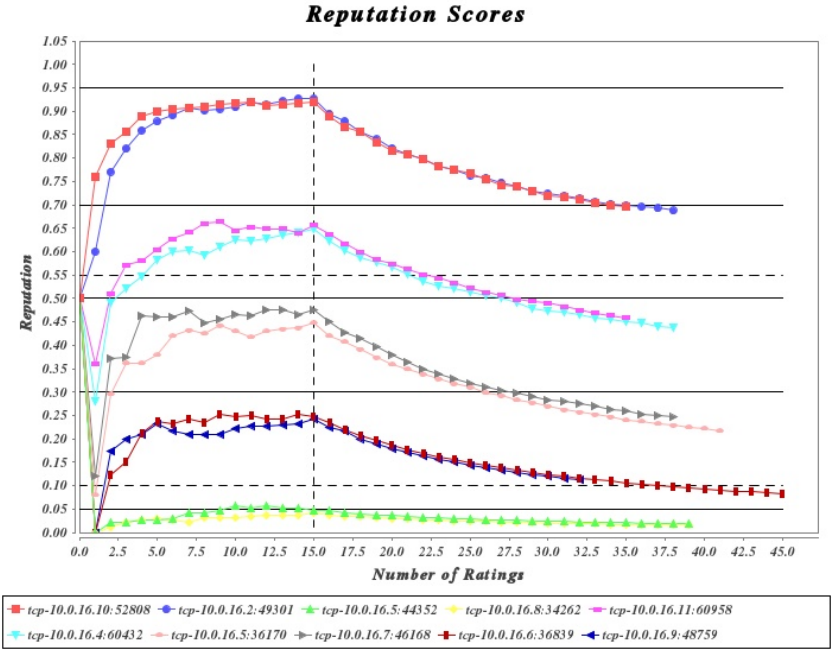


Figure 12: Trend of parties' reputation

analysis. Below we describe some of the graphs returned as output by NEVER (see Appendix B for further results).

The graph in Figure 12 shows the reputation trends of the ten parties calculated using the main model. The preset behaviour of each party is denoted by an horizontal line. The vertical dashed line denotes the time when parties change their behaviour. The horizontal dashed lines denote new parties' behaviours after the change. The trend of party's reputation is denoted by a polygonal line and each party is identified by its ip address and port number. Dashed lines are present only if a changeable scenario is set in the configuration file (parameter changeable_behaviour). Through this graph it is possible to analyse the evolution of parties' reputation in relation with the number of available ratings. The reputation

values shown are computed for the main trust model and are averaged over all runs. When changeable behaviour is set, it is also possible to analyse the reaction of the model when party's behaviour change in time. From Figure 12 we see that the ML model quite rapidly detects party's behaviour, with no significant error in reputation score assignation. Such model slowly adapts to party's new behaviour, that is when parties change their behaviour the ML model needs time to detect the new behaviour. According to other experiments, ML model adaptation to new party's behaviour is even worst when the change happens after a bigger number of interactions. Indeed, ratings about past interactions influence more the estimation of the actual behaviour. To reduce the effect of such estimation problem, the window and forgetting factor parameters should be used and tuned carefully.

Besides the main reputation model, parties's reputation is computed for each party with respect to all models specified in the model list within the configuration file. Figure 13 considers the case of a party with behaviour $\theta = 0.95$. Vertical and horizontal lines have the same meaning as in the previous figure. For each trust model is drawn a polygonal line denoting party's reputation. Through this graph it is possible to analyse how different trust models evaluate party's reputation and how they react in case of changeable behaviour. From this comparison it is possible to determine which is the best strategy for a given scenario. We can imagine scenarios where some kind of parties' behaviours are more probable than others, e.g. the majority of parties behave badly or goodly. Scenarios where the user is interested in preventing specific behaviours, e.g. parties try to reach a good reputation and then they start to behave badly. In Figure 13 it is possible to see how the six models react when party's behaviour changes from a behaviour $\theta = 0.95$ to a behaviour $\theta = 0.55$. From this graph we notice that the models using window parameter or forgetting factor are able to detect more rapidly new party's behaviour. Specifically, the models using the forgetting factor parameter adapt more rapidly than the ones using the window parameter. We notice also that ML models are more accurate in estimation than Beta models, the reputation values assigned in the average by ML models are closer to true

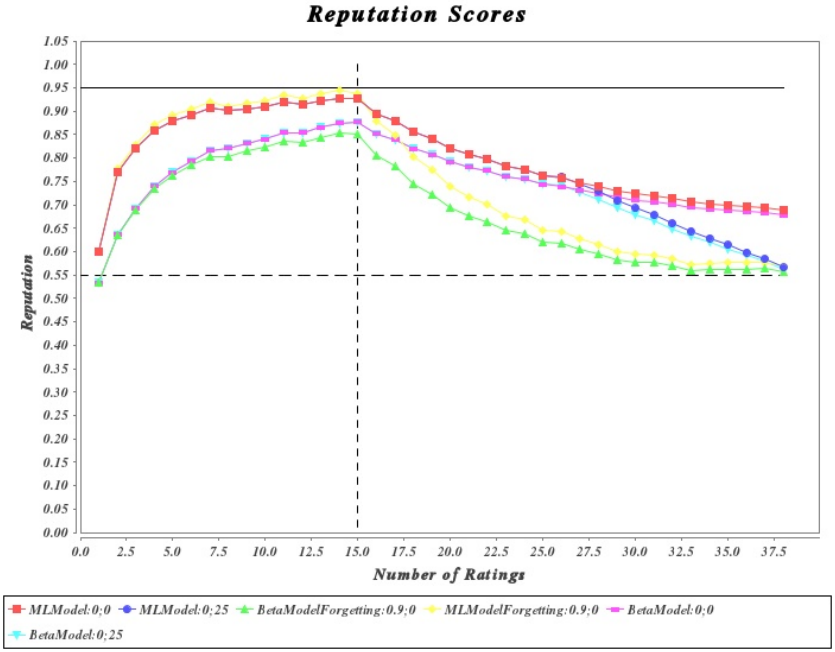


Figure 13: Trend of single party's reputation respect to the reputation models configured

party's behaviour than those assigned by Beta models. After the change, when a new behaviour is assumed by the party, it happens the opposite, Beta models estimate better than ML models. We recall that such results are reported for illustration purpose, indeed for an accurate analysis of the models we need more cases and more interactions outcomes.

When a system with several parties is set up, it becomes hard to read the graph reported in Figure 12. In order to manage systems with several parties, NEVER returns as output four different graphs where data is aggregated and parties are grouped depending on their behaviour. Parties are divide as follows:

- Group 1: parties whose behaviour θ is such that $0 \leq \theta < 0.25$;

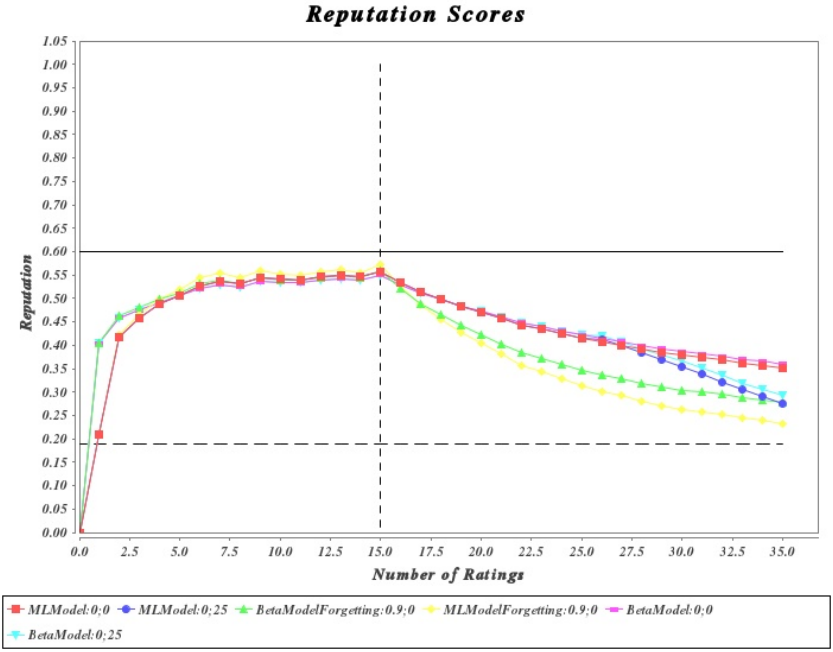


Figure 14: Trend of aggregated parties' reputation, Group 3

- Group 2: parties whose behaviour θ is such that $0.25 \leq \theta < 0.5$;
- Group 3: parties whose behaviour θ is such that $0.5 \leq \theta < 0.75$;
- Group 4: parties whose behaviour θ is such that $0.75 \leq \theta \leq 1$.

Figure 14 shows one of these graphs, where data about parties with a behaviour between $\theta = 0.5$ and $\theta < 0.75$ is aggregated. The horizontal line denotes the true behaviour of the group; such value is obtained through a weighted mean among parties' behaviour belonging to the group, i.e. the weight of each behaviour in the group is given by the number of parties having such behaviour. The same computation is done for the group's new behaviour after the change and the group's reputation. Such values are denoted by an horizontal dashed line and

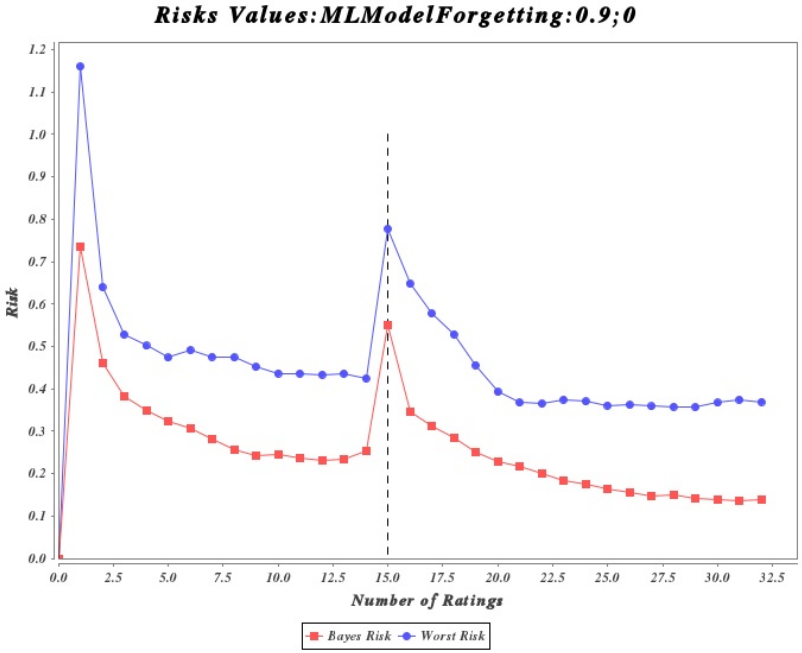


Figure 15: Risk trend for a single reputation model

a polygonal line (for each model) respectively. Notice that this kind of graphs, reporting aggregate data, are given as output with the aim of helping in graphically visualizing the trend of parties' reputation in the system. The groups' configuration is fixed and arbitrarily chosen, they have not a clear meaning for systems evaluation.

Finally, Figure 15 shows a graph where are reported the empirical values of the two risk functions presented in Section 3.2, namely bayes and worst risks. This graph is returned as output for each model specified in the configuration file. The two risks can be interpreted as follows: the *bayes risk* can be seen as the average risk in the system, where risk means the possibility of inferring party's behaviour wrongly; the *worst risk* is instead the risk incurred in the worst case, i.e. the maximum risk for a

given behaviour. We use these two risks for evaluating the goodness of the decision taken by reputation systems. They are a measure of the error committed by a system in estimating parties' reputation. We choose to use this two risk functions instead of more common error measures (e.g., absolute error), because we think they are more appropriate for the assessment of reputation systems, as discussed in Chapter 3.

Chapter 6

Concluding Remarks

In this chapter we propose a brief discussion on the main contributions of our work. Moreover we discuss some of the research literature related to the theoretical and software frameworks and tools proposed in the thesis. Finally, we present our main future research directions.

6.1 Discussion and Related Work

In this thesis, we addressed the issues related to the study, design and implementation of trust and reputation systems by proposing a methodology for their analysis and evaluation. Our proposal aims at providing theoretical and software frameworks and tools useful for the development and deployment of such systems. Three are the main contributions presented in the thesis: 1) a general framework for the assessment of trust and reputation models, 2) an analysis methodology for the design of such systems and 3) a tool for their evaluation in real networking infrastructure.

A Framework for the Assessment of Trust and Reputation Models

In Chapter 3 we have proposed a framework to analyze probabilistic trust systems based on bayesian decison theory. We have focused on probabilistic trust and reputation models, examining the behaviour of

two risk quantities: bayes and worst risks. Such quantities rely on the concept of loss function. Our results allow us to characterize the asymptotic behaviour of probabilistic trust systems. In particular, we have shown how to determine the limit value of both bayes and worst risks, and their exact exponential rates of convergence, in the case of independent and identically distributed observations. Specifically, we have shown that decision functions based on maximum likelihood and maximum a posteriori decision functions are asymptotically optimal. We have complemented these theoretical results with a set of numerical simulations. We have also considered the case where the raters may misbehave, arguing that a data-model with hidden variables is well-suited to model this kind of scenario; this naturally prompts the use of Expectation-Minimization algorithms to practically perform parameter estimation in this context.

Several studies seek to evaluate and compare trust and reputation systems. Despotovic and Aberer [DA04] propose a probabilistic approach, based on maximum likelihood estimation, for the assessment of peers' trustworthiness in P2P networks. They use simulations techniques for evaluating the quality of their proposal, by considering two settings relevant for P2P communities. In [XL04] Xiong and Liu present PeeTrust, a reputation-based trust supporting framework for P2P electronic communities. The feasibility, effectiveness, and benefits of their approach is demonstrated by results of simulation-based experiments. Boukerch et al. [BXEK07] present a trust and reputation management scheme for wireless sensor networks, that can protect the security of transacting entities. Trough experimental results they state that, due to the minimal overhead provided by their scheme, their approach can be adequately adopted for wireless sensor networks.

To the best of our knowledge, only a few studies follow an approach similar to the one we propose for investigating trust and reputation systems. Among these, Sassone et al. in [SKN07] propose a formal framework for the comparison of probabilistic trust models based on KL-divergence (see Section 2.2). In their work KL-divergence is used as a measure of the quality of reputation functions. In particular, in our

framework we use the KL-divergence as a possible loss functions definition. ElSalamouny et al., in [ESN10], introduce the foundations for the hidden Markov model based (HMM-based) trust model, analysing the case of ratees exhibiting dynamic behaviours. They perform an experimental comparison between HMM-based trust algorithm and the Beta-based trust algorithm with an exponential decay scheme. Our analysis allow us to characterize the asymptotic behaviour of probabilistic trust systems, an issue that is ignored both in [SKN07] and in [ESN10].

Analysis Methodology for the Design of Reputation Systems

In Chapter 4 we propose a verification methodology for trust and reputation models by means of a coordination language. By relying on KLAIM, we show how coordination languages and formal methods can be beneficial to the field of reputation systems. More specifically, we illustrate how such systems can be specified with KLAIM and analysed with KLAIM-based stochastic logic and model checker.

To the best of our knowledge, our contribution is the first study based on the use of a formal coordination language. Some related works make use of computational software programs or software for multi-agent modelling, for simulating their reputation models. Wang and Vassileva [WV03] propose a Bayesian network-based trust model, for building reputation based on recommendations in peer-to-peer networks. For evaluating their approach, they develop a simulation of a file sharing system in a peer-to-peer network. In [SVB04], Schlosser et al. present a formal model for describing metrics in reputation systems. By means of this model they evaluate the effectiveness of reputation systems on the basis of simulations. The authors use the RePast simulator [REP] to conduct various studies on reputation systems in a multi-agent context. Xiong and Liu [XL03] present a first attempt of an adaptive reputation-based trust model for P2P electronic communities. They evaluate the effectiveness and benefits of their trust model by experimental results obtained through simulations.

Such studies mainly differ from ours because we rely on formal methods' tools, such as coordination languages, stochastic modal logics, sim-

ulation and model checking, that permit expressing and evaluating performance measures in terms of logical formulae. This is a strong conceptual framework that provides abstraction primitives for conveniently specifying reputation systems and their properties. Stochastic modelling and verification tools permit considering the typical *uncertainty* of real systems. In this way, the analysis can cover more relevant situations of the considered system, when compared to an analysis based only on simulations.

NEVER

In Chapter 5 we presented NEVER, a network-aware tool for evaluating trust and reputation systems. The design of NEVER is based on the KLAIM formal specification of trust and reputation system presented in Chapter 4. We have used the Java library KLAVA for implementing the reputation models specified in KLAIM. NEVER allows users to rapid prototype and test reputation system models in a real network environment, thus realizing a generic testbed for evaluating trust and reputation systems. We have discussed the architecture of NEVER showing the logical structure and short part of its implementation. We have shown how NEVER works by means of experimental data obtained through the evaluation of some implemented models.

Among the many works in the literature, to the best of our knowledge, our contribution is the first effective tool allowing the evaluation of reputation systems in a real networked execution environment. Several works provide a testbed for the evaluation of trust and reputation systems, using simulation techniques. For example in [FKM+05], a simulator implemented in Java is proposed as testbed (the ART testbed) enabling a competition forum for evaluating trust systems. In this case, no networking or other real world aspects are taken into account. Other examples of testbed are TREET [KC10] and the one proposed in [IJZ12]. The latter testbed is used for the evaluation of robustness of reputation systems. Specifically, this proposal focuses on robustness against unfair ratings, i.e. against parties that release scores that intentionally under-estimate interaction outcomes. The TREET testbed is proposed as

an alternative to ART, which is considered not well-suited for general-purpose experimentation of reputation systems (it has, indeed, agents evaluation as its main purpose).

All these proposals are simulators or just architecture designs of testbeds that focus on marketplace applications. Our proposal, instead, does not fix a specific environment in which parties interact, but uses interactions as an abstraction of any parties relation. Moreover, we explicitly focus on probabilistic trust and reputation systems and on how they are evaluated. Our work aims at filling the gap between simulation and implementation of reputation systems, where networking aspects may play an important role when choosing and tuning reputation systems. Indeed such aspects must be considered when implementing these systems. Specifically, problems such as how to rate parties when interactions are affected by network delays, or how to rate parties that are sporadically connected, have to be addressed. For this reason, reputation systems in NEVER are specified so that such problems can be taken into account by users when evaluating the systems. Indeed, they can be tuned on the basis of the features of the underlying network infrastructure exploited by NEVER for the execution.

6.2 Further Research Directions

We intend to extend the framework presented in Chapter 3 to different data models, with rating values released in different ways. In particular, we would like to deepen our analysis of misbehaving raters. Another issue is how to evaluate the fitness of the model to the data actually available and, in general, how to assess the trade-off between tractability and accuracy of the model.

The verification approach presented in Chapter 4 can be extended to other reputation models proposed in the literature (e.g., those surveyed in [SS05; JIB07]). It would be also interesting to consider attacks to reputation systems, e.g. by cheating raters or through purchase of reviews. This would require the extension of our approach raising the issue of how to model these behaviours within our proposal.

Finally, concerning our tool NEVER presented in Chapter 5, we intend to extend our investigation to reputation systems over network architectures that rely on distributed rating servers, rather than on a single, centralised, one. Indeed, many authors have proposed adaptations of trust models for decentralised architectures. A reputation model adapted to ad-hoc networks for enhancing collaborations is proposed in [NCL07]. For evaluating the relationships among devices in pervasive computing environments, a trust management scheme is introduced in [DS08], while [AD01] presents data structures and algorithms for assessing trust in a peer-to-peer environment. We intend to study how different underlying network architectures affects the performances of a given reputation model.

It is our intention to extend NEVER to process real data from applications. The tool would be embedded in real applications and used to evaluate reputations systems in such environments. Applications could use reputation models in two different modalities: active or passive. In the active case, parties would compute reputation scores and use them to drive their interactions. In this modality the behaviour of an application would be modified by the deployed reputation system. In the passive case, the tool would collect rating values, compute reputation scores and just store them, without using such data to drive parties' interactions. The computed information would thus be used only for evaluating reputation systems. The passive modality would be useful in case of applications already deployed and in production. In this case it is important to understand how the application's behaviour would change before altering it. The passive modality could be also used for monitoring applications relying on existing reputation systems and contrast their reputation models with respect to the reputation models implemented in our tool.

Appendix A

Simulation Results

This appendix contains further simulation results of the StoKlaim specification, introduced in Chapter 4. Figures' captions and charts' legends show the relevant parameters of the experiment. A comment of the data can be found in Section 4.2.1.

The appendix is organized as follows: Section A.1 reports the charts relative to parties with no initials ratings; Section A.2 reports the charts relative to parties with four initials ratings, which fix parties' initial reputations to 0.5.

A.1 No initials ratings

All charts in this section are relative to the case of systems where parties have not initial ratings. Each chart reports the parties' reputation trends respect to the Beta model and ML model, and the estimation error committed by the models. The error is measured as the norm-1 distance (see Section 3.1.3) between probability distributions. The two distributions are the one determined by party's behaviour and the one determined by party's reputation. The charts in this section concern parties whose behaviour θ takes value in the set $\{0.1, 0.25, 0.3, 0.4, 0.5, 0.6, 0.7, 0.75, 0.8, 0.9, 1\}$.

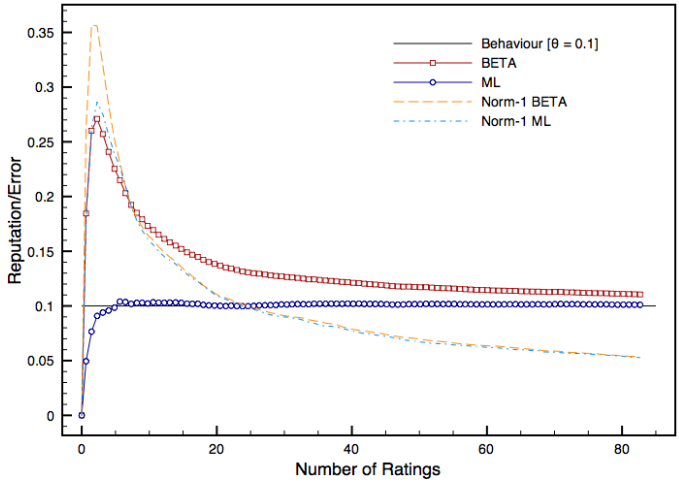


Figure 16: Reputation and error trends for parties with behaviour $\theta = 0.10$ and no initial ratings assigned

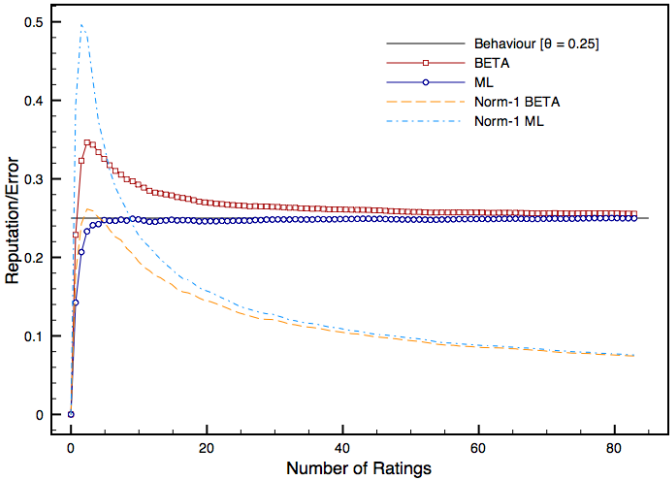


Figure 17: Reputation and error trends for parties with behaviour $\theta = 0.25$ and no initial ratings assigned

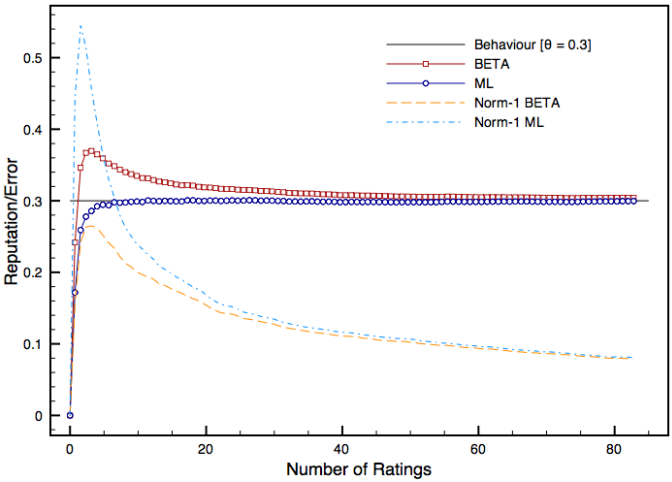


Figure 18: Reputation and error trends for parties with behaviour $\theta = 0.30$ and no initial ratings assigned

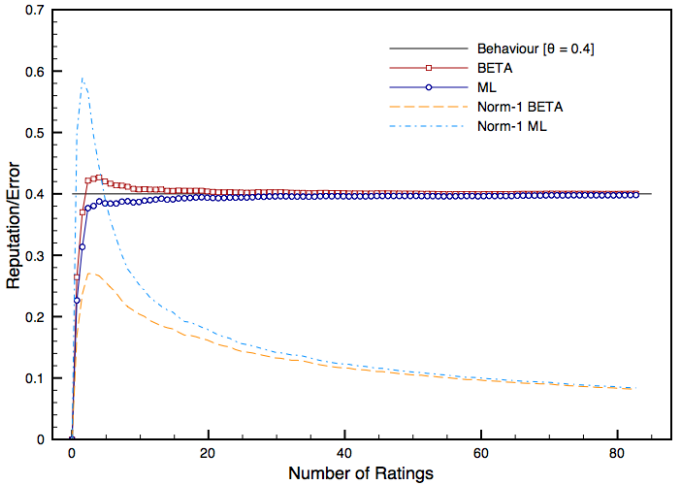


Figure 19: Reputation and error trends for parties with behaviour $\theta = 0.40$ and no initial ratings assigned

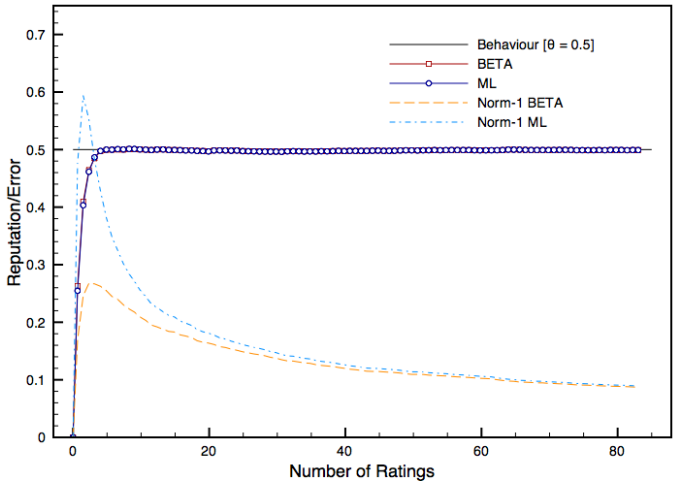


Figure 20: Reputation and error trends for parties with behaviour $\theta = 0.50$ and no initial ratings assigned

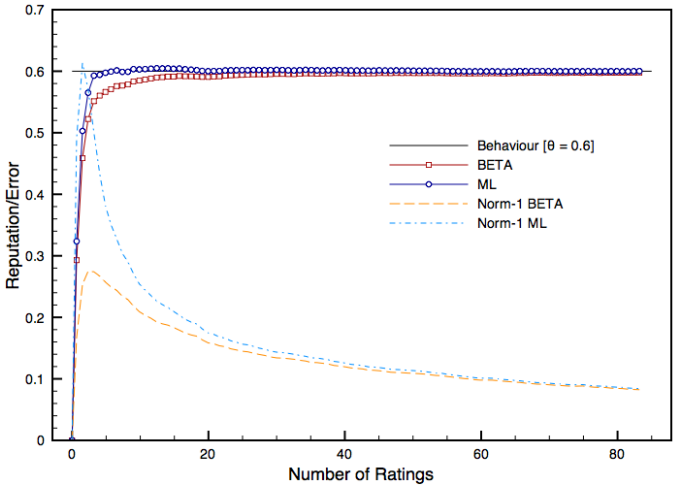


Figure 21: Reputation and error trends for parties with behaviour $\theta = 0.60$ and no initial ratings assigned

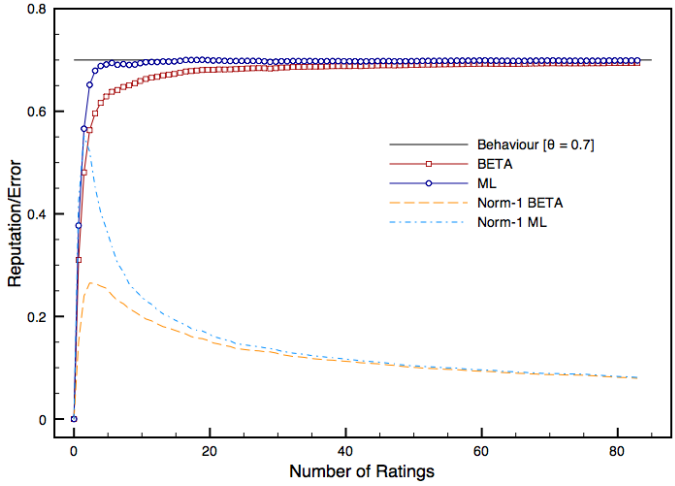


Figure 22: Reputation and error trends for parties with behaviour $\theta = 0.70$ and no initial ratings assigned

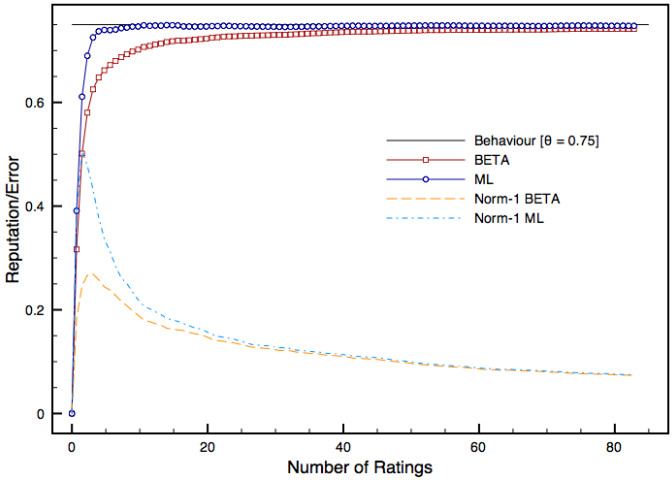


Figure 23: Reputation and error trends for parties with behaviour $\theta = 0.75$ and no initial ratings assigned

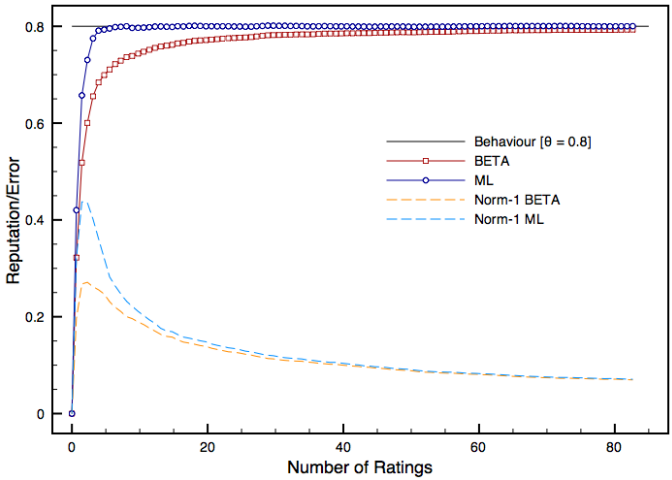


Figure 24: Reputation and error trends for parties with behaviour $\theta = 0.80$ and no initial ratings assigned

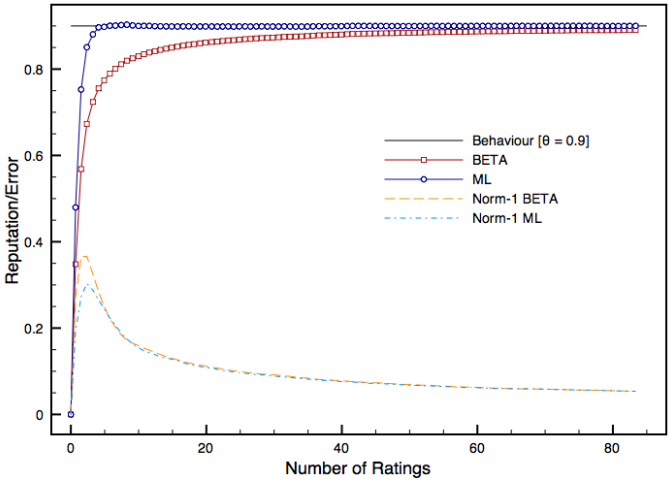


Figure 25: Reputation and error trends for parties with behaviour $\theta = 0.90$ and no initial ratings assigned

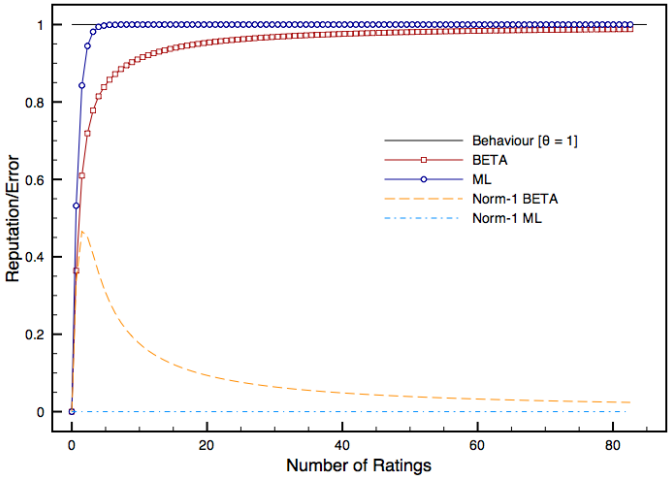


Figure 26: Reputation and error trends for parties with behaviour $\theta = 1$ and no initial ratings assigned

A.2 Four initials ratings

All charts in this section are relative to systems where parties have four initial ratings, which fix parties' initial reputation to 0.5. Each chart reports parties' reputation trends, with respect to the Beta model and ML model, and the estimation error committed by the models. The error is measured as the norm-1 distance (see Section 3.1.3) between probability distributions. One distribution is determined by party's behaviour, the other by party's reputation. The charts in this section concern parties whose behaviour θ takes value in the set $\{0, 0.25, 0.3, 0.4, 0.5, 0.6, 0.7, 0.75, 0.8, 0.9, 1\}$.

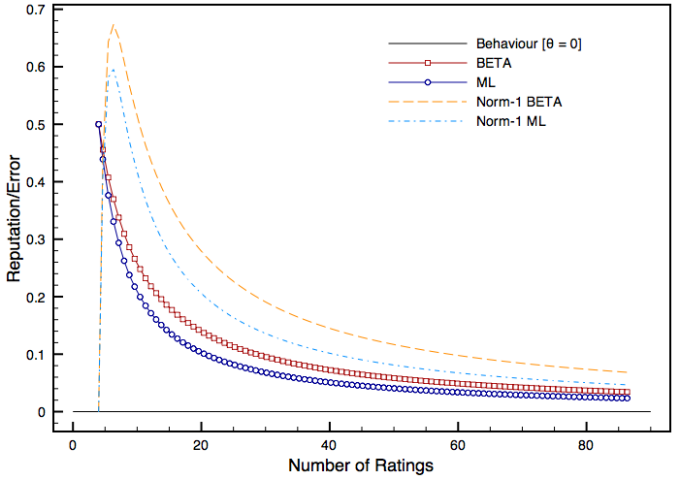


Figure 27: Reputation and error trends for parties with behaviour $\theta = 0$ and 4 initial ratings, which fix parties' initial reputation score to 0.5

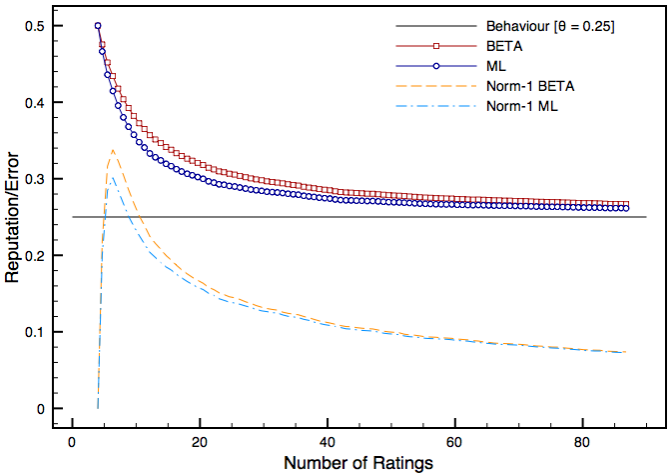


Figure 28: Reputation and error trends for parties with behaviour $\theta = 0.25$ and 4 initial ratings, which fix parties' initial reputation score to 0.5

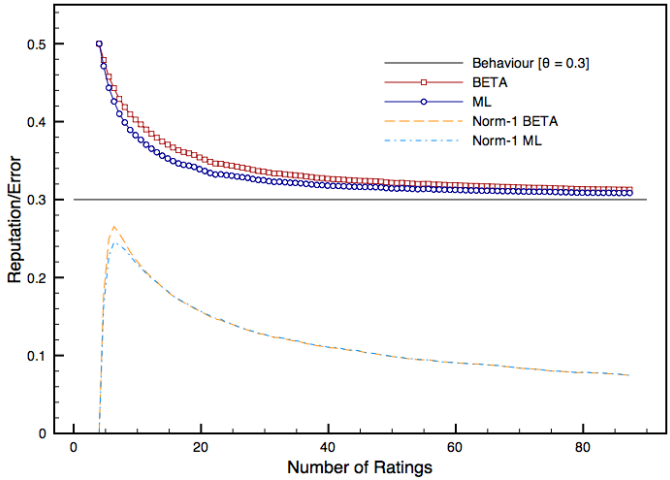


Figure 29: Reputation and error trends for parties with behaviour $\theta = 0.30$ and 4 initial ratings, which fix parties' initial reputation score to 0.5

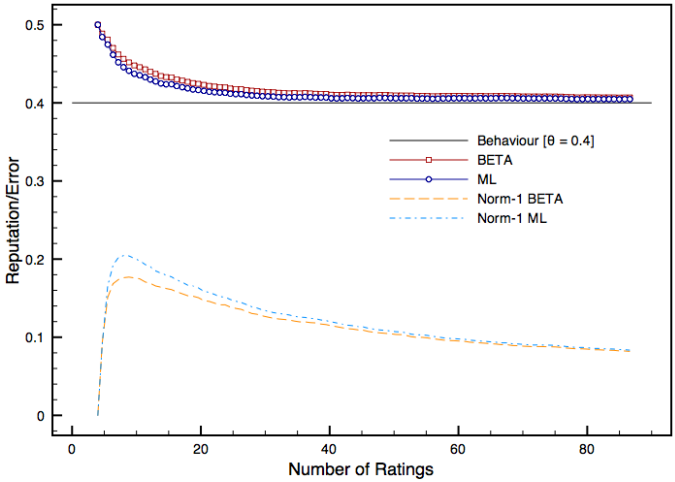


Figure 30: Reputation and error trends for parties with behaviour $\theta = 0.40$ and 4 initial ratings, which fix parties' initial reputation score to 0.5

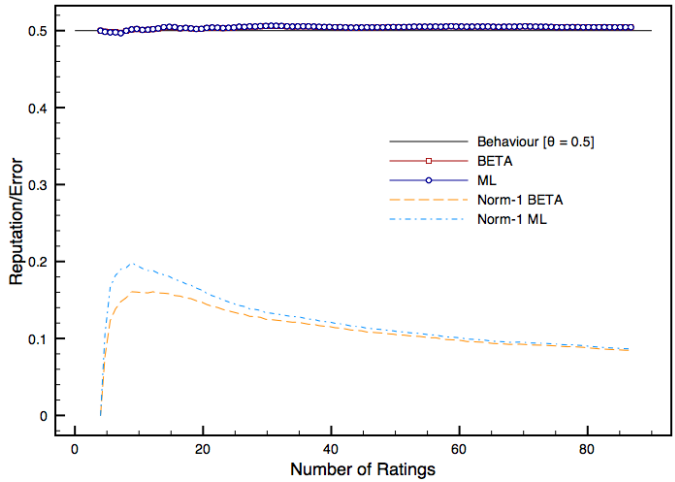


Figure 31: Reputation and error trends for parties with behaviour $\theta = 0.50$ and 4 initial ratings, which fix parties' initial reputation score to 0.5

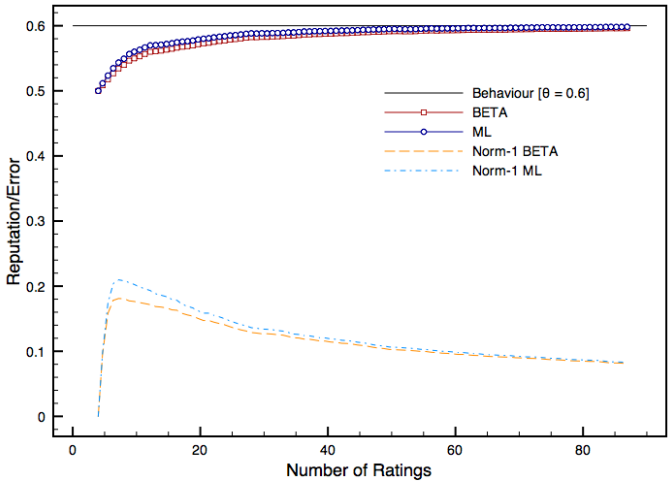


Figure 32: Reputation and error trends for parties with behaviour $\theta = 0.60$ and 4 initial ratings, which fix parties' initial reputation score to 0.5

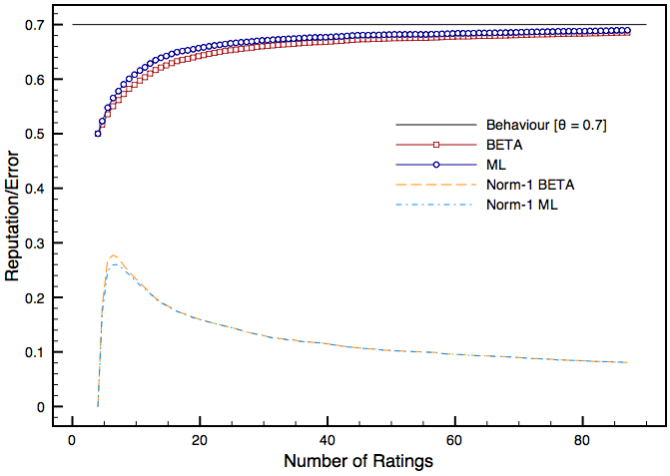


Figure 33: Reputation and error trends for parties with behaviour $\theta = 0.70$ and 4 initial ratings, which fix parties' initial reputation score to 0.5

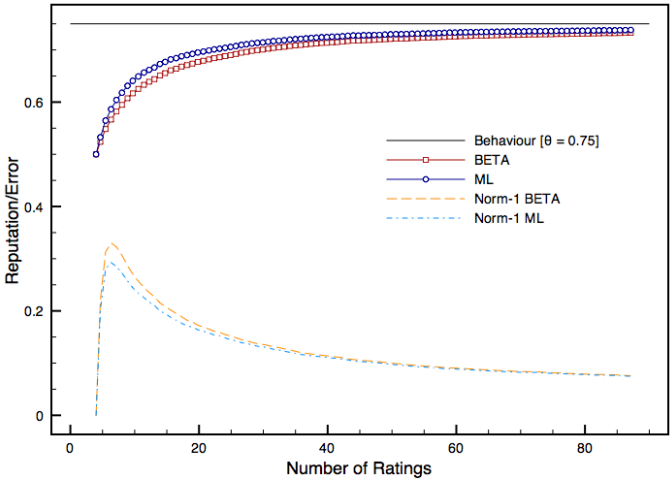


Figure 34: Reputation and error trends for parties with behaviour $\theta = 0.75$ and 4 initial ratings, which fix parties' initial reputation score to 0.5

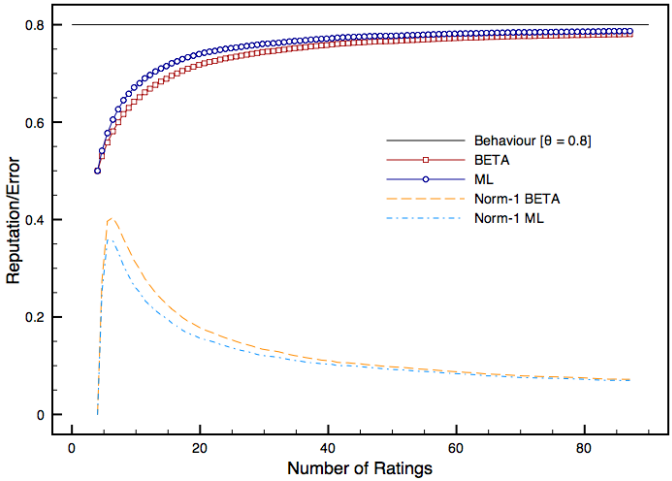


Figure 35: Reputation and error trends for parties with behaviour $\theta = 0.80$ and 4 initial ratings, which fix parties' initial reputation score to 0.5

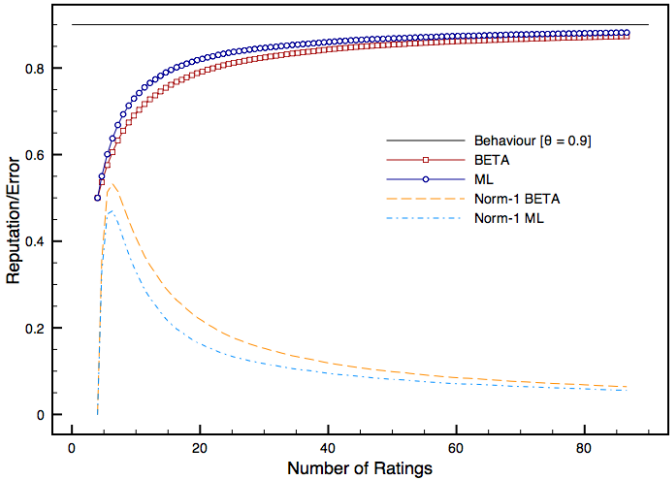


Figure 36: Reputation and error trends for parties with behaviour $\theta = 0.90$ and 4 initial ratings, which fix parties' initial reputation score to 0.5

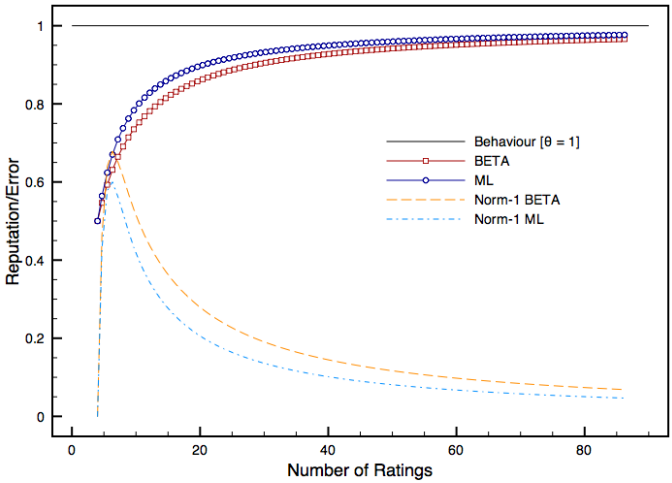


Figure 37: Reputation and error trends for parties with behaviour $\theta = 1$ and 4 initial ratings, which fix parties' initial reputation score to 0.5

Appendix B

NEVER Results

This appendix shows charts relative to the NEVER illustrative experiment, introduced in Chapter 5. Figures' captions and charts' legends report the relevant parameters of the experiment. A comment of the data can be found in Section 5.5.

The appendix is organized as follows: Section B.1 reports charts concerning the trend of groups' reputation score, presented in Section 5.5; Section B.2 reports charts concerning the reputation trend of the single parties active in the system, with respect to the models defined through the configuration file, the different models are reported in the charts' legends; Section B.3 reports the charts concerning the empirical values of the bayes and worst risks for each model configured.

B.1 Group Charts

All the charts in this section concern the trend of aggregate reputations of the group presented in Section 5.5. Parties are divided in group as follow:

- Group 1: parties whose behaviour θ is such that $0 \leq \theta < 0.25$;
- Group 2: parties whose behaviour θ is such that $0.25 \leq \theta < 0.5$;
- Group 3: parties whose behaviour θ is such that $0.5 \leq \theta < 0.75$;
- Group 4: parties whose behaviour θ is such that $0.75 \leq \theta \leq 1$.

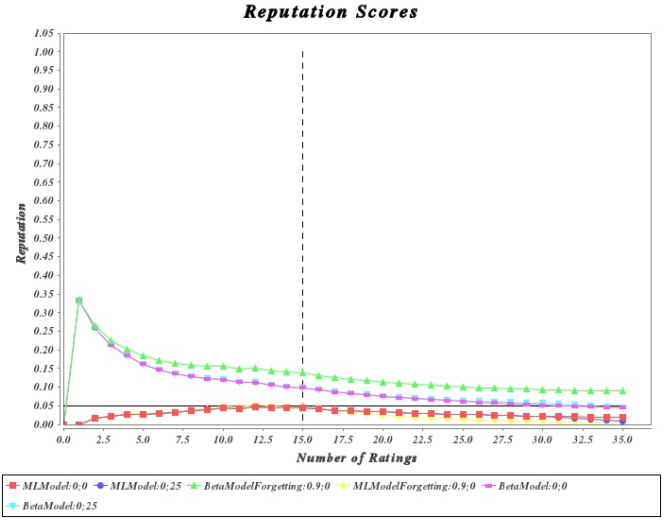


Figure 38: Trend of aggregated parties' reputation, Group 1

Figure 39: Trend of aggregated parties' reputation, Group 2

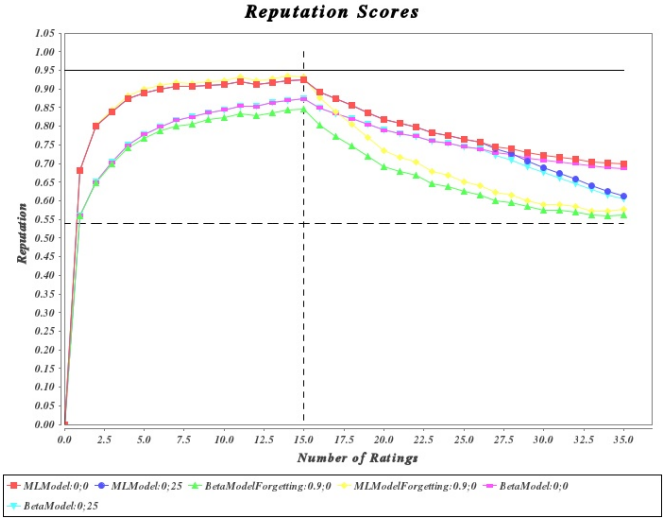


Figure 40: Trend of aggregated parties' reputation, Group 4

B.2 User Charts

All the charts in this section show the reputation trend of a single party active in the system, with respect to the model set in the configuration file. Each chart reports the trends of all models configured. Party's behaviour θ takes value in the set $\{0.05, 0.3, 0.5, 0.7, 0.95\}$

Figure 41: Trend of a single party's reputation, whose behaviour is $\theta = 0.95$, with respect to the reputation models configured

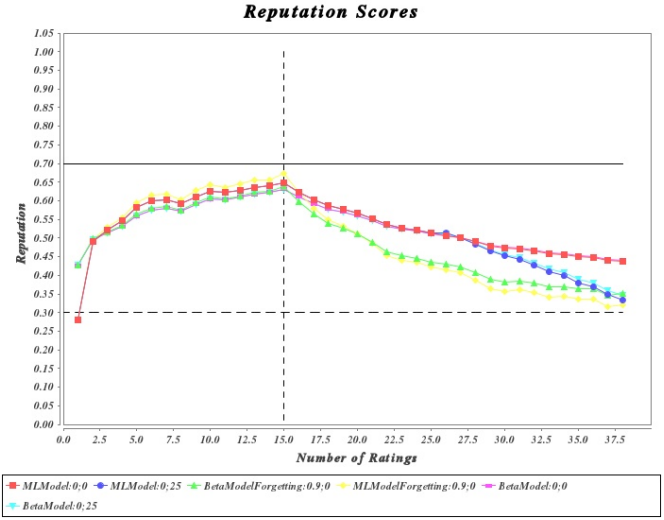


Figure 42: Trend of a single party's reputation, whose behaviour is $\theta = 0.7$, with respect to the reputation models configured

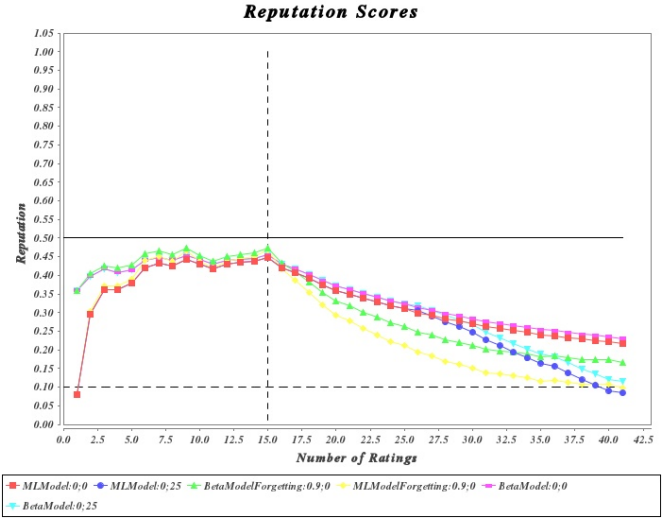


Figure 43: Trend of a single party's reputation, whose behaviour is $\theta = 0.5$, with respect to the reputation models configured

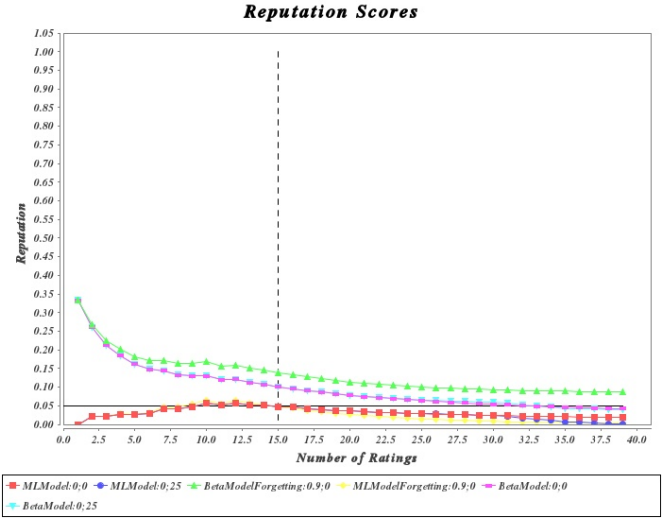


Figure 44: Trend of a single party's reputation, whose behaviour is $\theta = 0.05$, with respect to the reputation models configured

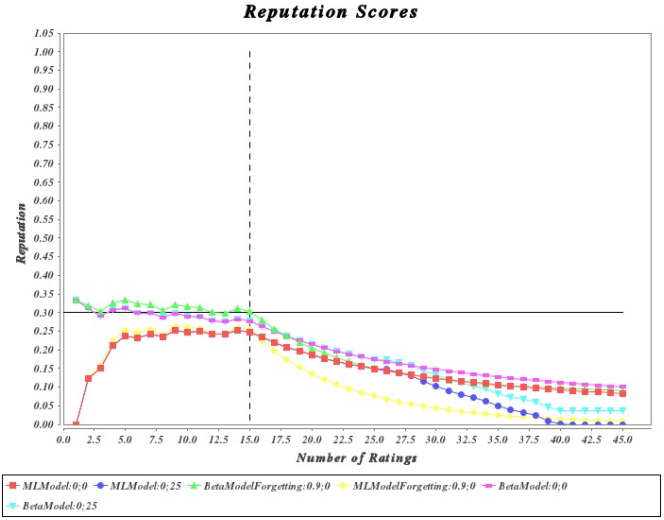


Figure 45: Trend of a single party's reputation, whose behaviour is $\theta = 0.3$, with respect to the reputation models configured

B.3 Charts of the Models

For each configured model, the charts in this section show the trend of the bayes and worst risks empirical values. The models set in our experiment were the following:

- ML model without window and forgetting factor;
- ML model with window=25 and without forgetting factor;
- ML model without window and with forgetting factor=0.9;
- Beta model without window and forgetting factor;
- Beta model with window=25 and without forgetting factor;
- Beta model without window and with forgetting factor=0.9.

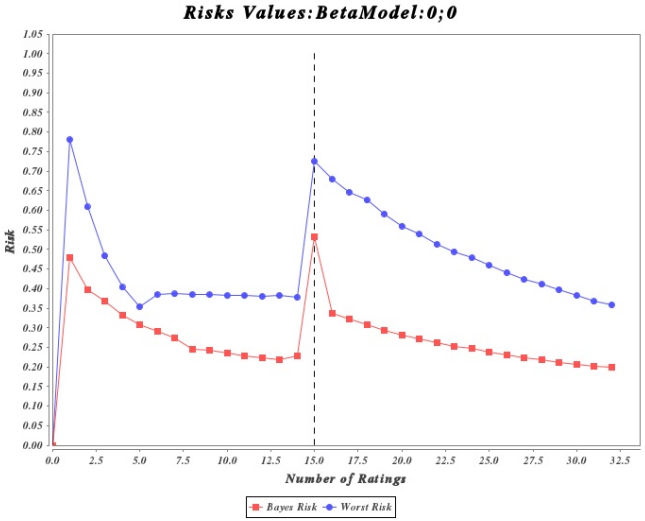


Figure 46: Risk trend for a single reputation model

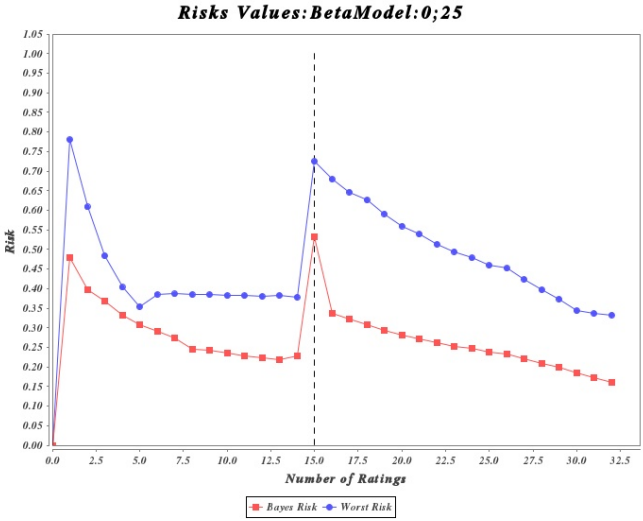


Figure 47: Risk trend for a single reputation model

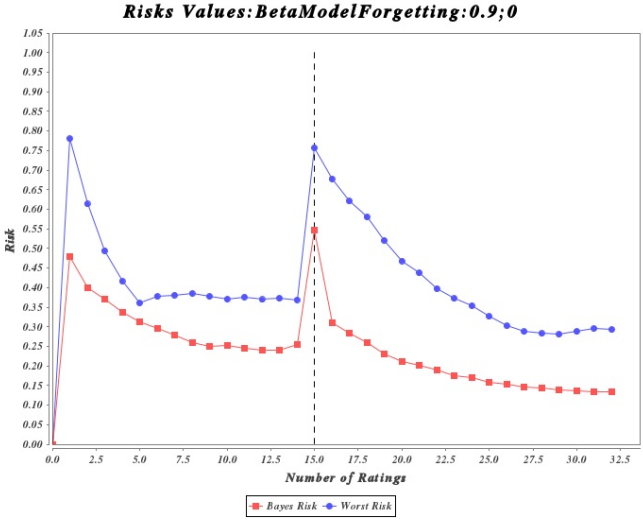


Figure 48: Risk trend for a single reputation model

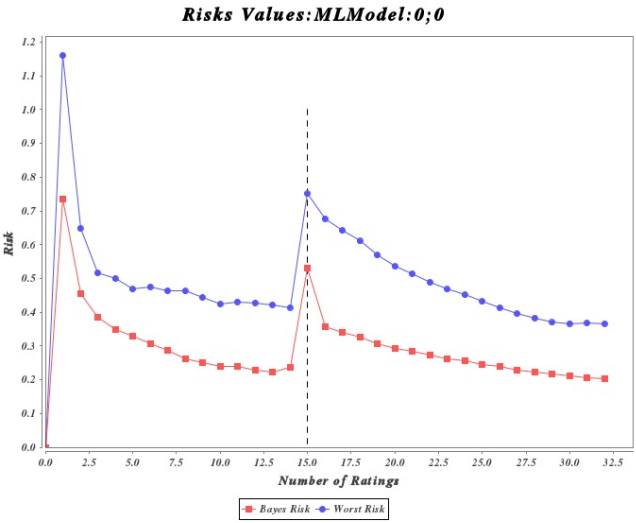


Figure 49: Risk trend for a single reputation model

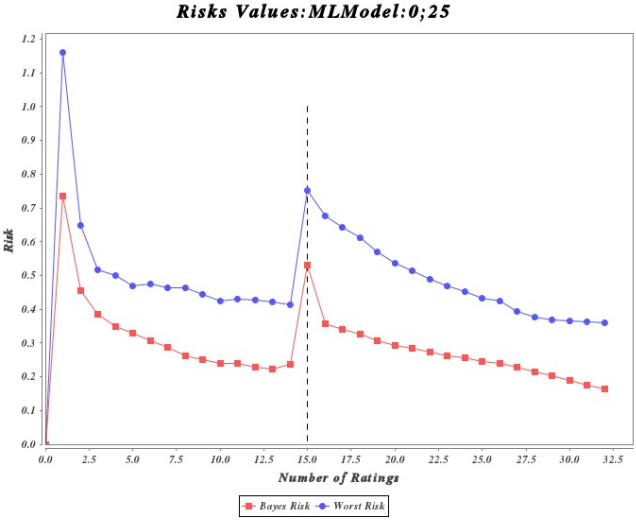


Figure 50: Risk trend for a single reputation model

References

[AD01] Karl Aberer and Zoran Despotovic. Managing trust in a peer-2-peer information system. In *the tenth international conference on Information and knowledge management*, pages 310–317. ACM, 2001.

[ARH00] Alfarez Abdul-Rahman and Stephen Hailes. Supporting trust in virtual communities. In *the 33rd Annual Hawaii International Conference on System Sciences, 2000.*, pages 9–pp. IEEE, 2000.

[ASSB00] Adnan Aziz, Kumud Sanwal, Vigyan Singhal, and Robert Brayton. Model-checking continuous-time markov chains. *ACM Transactions on Computational Logic (TOCL)*, 1(1):162–170, 2000.

[Bar12] David Barber. *Bayesian Reasoning and Machine Learning*. Cambridge University Press, 2012.

[BBD+03] Lorenzo Bettini, Viviana Bono, Rocco De Nicola, Gianluigi Ferrari, Daniele Gorla, Michele Loreti, Eugenio Moggi, Rosario Pugliese, Emilio Tuosto, and Betti Venneri. The klaim project: Theory and practice. *Global Computing. Programming Environments, Languages, Security, and Analysis of Systems*, pages 88–150, 2003.

[BC13] Michele Boreale and Alessandro Celestini. Asymptotic risk analysis for trust and reputation systems. In *SOFSEM 2013: Theory and Practice of Computer Science*, volume 7741 of *Lecture Notes in Computer Science*, pages 169–181. Springer Berlin Heidelberg, 2013.

[BDF+05] Lorenzo Bettini, Rocco De Nicola, Daniele Falassi, Marc Lacoste, and Michele Loreti. A flexible and modular framework for implementing infrastructures for global computing. In *Distributed Applications and Interoperable Systems*, pages 1097–1119. Springer, 2005.

[BDP02] Lorenzo Bettini, Rocco De Nicola, and Rosario Pugliese. Klava: a java package for distributed and mobile applications. *Software: Practice and Experience*, 32(14):1365–1394, 2002.

[Ber85] James O. Berger. *Statistical Decision Theory and Bayesian Analysis*. Springer, second edition edition, 1985.

[BFIK99] Matt Blaze, Joan Feigenbaum, John Ioannidis, and AngelosD. Keromytis. The role of trust management in distributed systems security. In *Secure Internet Programming*, volume 1603 of *Lecture Notes in Computer Science*, pages 185–210. Springer Berlin Heidelberg, 1999.

[BKH99] Christel Baier, Joost-Pieter Katoen, and Holger Hermanns. Approximative symbolic model checking of continuous-time markov chains. In *CONCUR99 Concurrency Theory*, pages 146–161. Springer, 1999.

[BPP11] Michele Boreale, Francesca Pampaloni, and Michela Paolini. Quantitative information flow, with a view. In *16th European conference on Research in computer security*, ESORICS'11, pages 588–606, 2011.

[BXEK07] Azzadine Boukerche, Li Xu, and Khalil El-Khatib. Trust-based security for wireless ad hoc and sensor networks. *Computer Communications*, 30(11):2413–2427, 2007.

[CDT13a] Alessandro Celestini, Rocco De Nicola, and Francesco Tiezzi. Network-aware evaluation environment for reputation systems. In *Trust Management VII – 7th IFIP WG 11.11 International Conference, IFIPTM (to appear)*, 2013.

[CDT13b] Alessandro Celestini, Rocco De Nicola, and Francesco Tiezzi. Specifying and analysing reputation systems with a coordination language. In *the 28th Annual ACM Symposium on Applied Computing (SAC'13)*. ACM, 2013.

[CL10] Francesco Calzolai and Michele Loreti. Simulation and analysis of distributed systems in klaim. In *Coordination Models and Languages*, pages 122–136. Springer, 2010.

[CT06] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. John Wiley & Sons, second edition edition, 2006.

[DA04] Zoran Despotovic and Karl Aberer. A probabilistic approach to predict peers performance in p2p networks. *Cooperative Information Agents VIII*, pages 62–76, 2004.

[DFP98] Rocco De Nicola, Gian Luigi Ferrari, and Rosario Pugliese. Klaim: A kernel language for agents interaction and mobility. *Transaction on Software Engineering*, 24(5):315–330, 1998.

[DKL+06a] Rocco De Nicola, Joost-Pieter Katoen, Diego Latella, Michele Loreti, and Mieke Massink. Klaim and its stochastic semantics. Technical report, 2006.

[DKL+06b] Rocco De Nicola, Joost-Pieter Katoen, Diego Latella, Michele Loreti, and Mieke Massink. MoSL: A Stochastic Logic for StoKlaim. Technical Report ISTI-06-35, 2006.

[DKL+07] Rocco De Nicola, Joost-Pieter Katoen, Diego Latella, Michele Loreti, and Mieke Massink. Model checking mobile stochastic logic. *Theoretical Computer Science*, 382(1):42–70, 2007.

[DL05] Rocco De Nicola and Michele Loreti. Momo: A modal logic for reasoning about mobility. In *Formal Methods for Components and Objects*, pages 95–119. Springer, 2005.

[DLM05] Rocco De Nicola, Diego Latella, and Mieke Massink. Formal modeling and quantitative analysis of klaim-based mobile systems. In *the 20th Annual ACM Symposium on Applied Computing*, pages 428–435. ACM, 2005.

[DS08] Mieso K Deno and Tao Sun. Probabilistic trust management in pervasive computing. In *IEEE/IFIP International Conference on Embedded and Ubiquitous Computing, 2008. EUC'08.*, volume 2, pages 610–615. IEEE, 2008.

[EC82] E Allen Emerson and Edmund M Clarke. Using branching time temporal logic to synthesize synchronization skeletons. *Science of Computer programming*, 2(3):241–266, 1982.

[EFL+99] C Ellison, B Frantz, B Lampson, R Rivest, B Thomas, and T Ylonen. Spki certificate theory, rfc2693. *IETF SPKI Working Group*, 1999.

[EKS09] Ehab ElSalamouny, Karl Tikjøb Krukow, and Vladimiro Sassone. An analysis of the exponential decay principle in probabilistic trust models. *Theoretical Computer Science*, 410(41):4067–4084, 2009.

[ESN10] Ehab ElSalamouny, Vladimiro Sassone, and Mogens Nielsen. Hmm-based trust model. In *Formal Aspects in Security and Trust*, volume 5983, pages 21–35. Springer, 2010.

[FKM+05] Karen K Fullam, Tomas B Klos, Guillaume Muller, Jordi Sabater, Andreas Schlosser, Zvi Topol, K Suzanne Barber, Jeffrey S Rosenschein, Laurent Vercouter, and Marco Voss. A specification of the agent reputation and trust (art) testbed: experimentation and competition for trust in agent societies. In *the fourth international joint conference*

on Autonomous agents and multiagent systems, pages 512–518. ACM, 2005.

[Gmb12] Zimory GmbH. Zimory Enterprise Cloud, 2012. Web site: `http://www.zimory.de`.

[GS01] Geoffrey R. Grimmett and David R. Stirzaker. *Probability and Random Processes*. Oxford University Press, third edition edition, 2001.

[Hei] Gregor Heinrich. Parameter estimation for text analysis.

[IJZ12] Athirai Aravazhi Irissappane, Siwei Jiang, and Jie Zhang. Towards a comprehensive testbed to evaluate the robustness of reputation systems against unfair rating attacks. In *UMAP Workshops*, volume 12, 2012.

[JH07] Audun Jøsang and Jochen Haller. Dirichlet reputation systems. In *The Second International Conference on Availability, Reliability and Security, 2007. ARES 2007.*, pages 112–119. IEEE, 2007.

[JI02] Audun Jøsang and Roslan Ismail. The beta reputation system. In *the 15th bled electronic commerce conference*, pages 41–55, 2002.

[JIB07] Audun Jøsang, Roslan Ismail, and Colin Boyd. A survey of trust and reputation systems for online service provision. *Decision Support Systems*, 43(2):618 – 644, 2007.

[Kal02] Olav Kallenberg. *Foundations of Modern Probability*. Springer, 2002.

[KC10] Reid Kerr and Robin Cohen. Treet: the trust and reputation experimentation and evaluation testbed. *Electronic Commerce Research*, 10(3-4):271–290, 2010.

[KNS08] Karl Krukow, Mogens Nielsen, and Vladimiro Sassone. Trust models in ubiquitous computing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 366(1881):3781–3793, 2008.

[KPS02] Charlie Kaufman, Radia Perlman, and Mike Speciner. *Network security: private communication in a public world*. Prentice Hall Press, 2002.

[Kul96] Rudolf Kulhavỳ. A kullback-leibler distance approach to system identification. *Annual Reviews in Control*, 20:119–130, 1996.

[LJ97] Charles C. Leang and Don H. Johnson. On the asymptotics of m-hypothesis bayesian detection. *IEEE Transactions on Information Theory*, 43(1):280 –282, jan 1997.

[LJHG11] Simon Lacoste-Julien, Ferenc Huszár, and Zoubin Ghahramani. Approximate inference for the loss-calibrated bayesian. 2011.

[LLYY09] Gehao Lu, Joan Lu, Shaowen Yao, and Yau Jim Yip. A review on computational trust models for multi-agent systems. *The Open Information Science Journal*, 2:18–25, March 2009.

[Lor10] Michele Loreti. SAM: Stochastic Analyser for Mobility, 2010. Available at `http://rap.dsi.unifi.it/SAM/`.

[MGM06] Sergio Marti and Hector Garcia-Molina. Taxonomy of trust: Categorizing p2p reputation systems. *Computer Networks*, 50(4):472–484, 2006.

[MMH02] Lik Mui, Mojdeh Mohtashemi, and Ari Halberstadt. A computational model of trust and reputation. In *the 35th Annual Hawaii International Conference on System Sciences, 2002. HICSS.*, pages 2431–2439. IEEE, 2002.

[NCL07] Chung Tien Nguyen, Olivier Camp, and Stephane Loiseau. A bayesian network based trust model for improving collaboration in mobile ad hoc networks. In *IEEE International Conference on Research, Innovation and Vision for the Future, 2007*, pages 144–151. IEEE, 2007.

[NIST09] A Survey of Access Control Models, National Institute of Standards and Technology (NIST), Working Draft. `http://csrc.nist.gov/news_events/privilege-management-workshop/PvM-Model-Survey-Aug26-2009.pdf`. 26 August 2009.

[NT94] B. Clifford Neuman and Theodore Ts'o. Kerberos: an authentication service for computer networks. *Communications Magazine, IEEE*, 32(9):33–38, 1994.

[OV11] Andrea Omicini and Mirko Viroli. Coordination models and languages: From parallel computing to self-organisation. *The Knowledge Engineering Review*, 26(01):53–59, 2011.

[RCD01] Davide Rossi, Giacomo Cabri, and Enrico Denti. Tuple-based technologies for coordination. In *Coordination of Internet Agents: Models, Technologies, and Applications*, pages 83–109. 2001.

[REP] RePast. Web site: `http://repast.sourceforge.net`.

[RJ96] Lars Rasmusson and Sverker Jansson. Simulated social control for secure internet commerce. In *the 1996 workshop on New security paradigms*, NSPW '96, pages 18–25. ACM, 1996.

[Rob07] Christian P. Robert. *The Bayesian Choice: From Decision-Theoretic Foundations to Computational Implementation*. Springer, second edition edition, 2007.

[SKN07] Vladimiro Sassone, Karl Krukow, and Mogens Nielsen. Towards a formal framework for computational trust. In *Formal Methods for Components and Objects*, volume 4709 of *Lecture Notes in Computer Science*, pages 175–184. Springer Berlin Heidelberg, 2007.

[SS01] Jordi Sabater and Carles Sierra. Regret: reputation in gregarious societies. In *the fifth international conference on Autonomous agents*, pages 194–195. ACM, 2001.

[SS05] Jordi Sabater and Carles Sierra. Review on computational trust and reputation models. *Artificial Intelligence Review*, 24:33–60, 2005.

[Sti99] David R. Stirzaker. *Probability and Random Variables: A Beginner's Guide*. Cambridge University Press, 1999.

[SVB04] Andreas Schlosser, Marco Voss, and Lars Bruckner. Comparing and evaluating metrics for reputation systems by simulation. In *A Workshop on Reputation in Agent Societies*, 2004.

[TPJL06] W T Luke Teacy, Jigar Patel, Nicholas R Jennings, and Michael Luck. Travos: Trust and reputation in the context of inaccurate information sources. *Autonomous Agents and Multi-Agent Systems*, 12(2):183–198, 2006.

[TW10] Andrew S. Tanenbaum and David J. Wetherall. *Computer Networks*. Prentice Hall, 2010.

[Wil91] David Williams. *Probability with Martingales*. Cambridge University Press, 1991.

[WV03] Yao Wang and Julita Vassileva. Trust and reputation model in peer-to-peer networks. In *Third International Conference on Peer-to-Peer Computing, 2003. (P2P 2003).*, pages 150–157. IEEE, 2003.

[XACML] eXtensible Access Control Markup Language (XACML) Version 3.0., 22 January 2013. OASIS Standard. `http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html`.

[XL03] Li Xiong and Ling Liu. A reputation-based trust model for peer-to-peer e-commerce communities. In *IEEE International Conference on E-Commerce, 2003. CEC 2003.*, pages 275–284. IEEE, 2003.

[XL04] Li Xiong and Ling Liu. Peertrust: Supporting reputation-based trust for peer-to-peer electronic communities. *IEEE Transactions on Knowledge and Data Engineering*, 16(7):843–857, 2004.

[ZM00] Giorgos Zacharia and Pattie Maes. Trust management through reputation mechanisms. *Applied Artificial Intelligence*, 14(9):881–907, 2000.