PhD-FSTC-9-2010
Faculty of Sciences, Technology and Communication

DISSERTATION

Defense held on 16 April 2010 in Luxembourg
in candidature for the degree of

DOCTOR OF THE UNIVERSITY OF LUXEMBOURG IN COMPUTER SCIENCE

by

EUGEN STAAB
Born on 2 June 1981 in Merzig, Germany

RELIABLE INFORMATION ACQUISITION IN THE PRESENCE OF MALICIOUS SOURCES

Dissertation defense committee

Dr. Claudia Eckert
Professor, Technische Universität München, Germany

Dr. Thomas Engel, Dissertation Supervisor
Professor, University of Luxembourg, Luxembourg

Dr. Guillaume Muller, Vice Chairman
CEA Saclay, France

Dr. Munindar P. Singh
Professor, North Carolina State University, USA

Dr. Leon van der Torre, Chairman
Professor, University of Luxembourg, Luxembourg

Abstract

In distributed systems in which autonomous entities exchange information with each other, these entities have the freedom to provide incorrect information. This becomes especially relevant in scenarios where entities have incentives to do so, e.g., in *peer-to-peer networks* or *volunteer computing systems*. Cryptographic mechanisms can help to ensure data integrity and authenticity. However, while these mechanisms achieve a reliable transmission of information, they do not prevent the creation of incorrect information in the first place. Therefore, additional mechanisms are necessary to enable an entity to assess the correctness of acquired information.

In specific applications, the correctness of acquired information can be verified, or be assessed by means of plausibility checks. This is not possible in general, though. Often the explicit verification of information is infeasible, for instance due to high costs. In this thesis, we investigate different generic approaches to ensure the correctness of information without explicitly verifying it. More precisely, we propose three mechanisms, each of which is suitable for certain scenarios and tasks: a *spot-checking mechanism* that uses known facts to ensure the correctness of acquired information; an *evidence-based trust model* that learns – based on past experience – to exclusively select trustworthy sources; and a *collusion detection algorithm* that addresses the main threat to mechanisms that ensure the correctness of information by means of redundancy. The proposed mechanisms are validated, theoretically and/or experimentally, against plausible attack strategies of malicious sources. We show that our mechanisms are able to deal with different kinds of attack strategies, and that our trust model and the collusion detection algorithm successfully identify malicious sources.

Keywords: Information Science, Artificial Intelligence, Sabotage Tolerance, Computational Trust, Volunteer Computing.

Acknowledgments

First and foremost, I would like to thank my supervisor Thomas Engel, for enabling me to do my PhD in his group, and for his great support throughout my studies. I am also very thankful to Guillaume Muller and Leon van der Torre, further members of my PhD Committee, for their excellent supervision. Moreover, I want to thank Leon van der Torre and Uli Sorger for invaluable discussions in an important phase of my work. I am very grateful to Claudia Eckert and Munindar P. Singh for being members of my defense committee. I want to thank all the researchers, with whom I have collaborated, to whom I owe inspiration, and from whom I have learned a lot. Especially, I want to mention Volker Fusenig, Martin Rehák and Martin Caminada. Furthermore, I want to thank Dominic Dunlop for revising my thesis and making helpful observations. My thanks go to all members of the SECAN-Lab and the ICR group, who let me have a great time, and have been very inspiring to me. I am thankful to all the friends that supported me during my PhD. Last but not least, I want to thank our secretaries, who were always helpful, even when they were overloaded with work.

I offer heartfelt thanks to my family and Astrid for their ceaseless support.

Contents

Chapter 1

Introduction

Enabled by the Internet, computing systems are becoming more and more distributed, and the entities taking part in the systems more and more autonomous. The correctness of exchanged information becomes an important issue for these systems. Faulty entities, and also entities with malicious intent, can provide incorrect information that has to be detected.

In computer networks, cryptographic mechanisms can ensure the authenticity and integrity of information that is sent from a node A to a node B. However, if A is not only the sender, but also the creator of the information, these mechanisms can generally not be used to ensure that A sends the *correct* information. Given the possibility that the sender has incentives to provide incorrect information, the receiver needs to *verify* the information. Unfortunately, in many distributed systems, an explicit verification of received information is too time-consuming, not feasible due to limited knowledge, or too costly. For instance, the latter is the case in *volunteer computing systems* (e.g., SETI@home [11]), a specific form of distributed computing:

> *In volunteer computing, a central server, the "master", assigns small computational tasks to clients, the "workers", which perform the computations and return their results to the master. Workers are private PCs that have been made available by their human owners, the* volunteers*. These volunteers want to contribute to the project for which the computations are performed; an example for such a project is the study of protein folding and related diseases [2]. The master has no control over the workers and so a worker's code can be patched by a volunteer. As a consequence, workers might return incorrect results. This actually happened in practice due to programming errors in the patches [105], but also due to malicious intent [10]. Therefore, results have to be verified by the master. However, there is no efficient verification scheme for general computational tasks [65]. Also, the verification of a result through recomputation and comparison is not possible, because it is as costly as the computation of the result itself.*

For this reason, mechanisms are required to *ensure the correctness of received information without explicitly verifying it*. There are two basic generic approaches to this problem: *spot-checking* and *replication*. In the first approach, as before, node A requests information from source B, and randomly checks some pieces of this information. Based on how correct the checked pieces of information are found to be, node A can attain probabilistic estimates of the correctness of the remaining information. In the second approach, that is replication, node A requests the same information from several randomly picked nodes $B_1, \ldots, B_n$. Assuming *independence* between these nodes and a low number of malicious nodes, the information that is returned by a majority or plurality of the nodes can be assumed to be correct.

The *first* goal of this thesis is to further the research on these two approaches. To this end, we first propose a mechanism that is based on the idea of spot-checking. The mechanism uses prepared challenges to check information sources, in particular in scenarios where even honest sources report information with an error rate greater than zero. In the case of the second idea (replication), we ask the question of how the independence assumption can be relaxed. For this purpose, we propose a mechanism that detects colluding behavior between information sources.

In addition to spot-checking and replication, humans use a very effective third approach, namely the *trust* they have in an information source. Once a human trusts in an information source, he assumes that information coming from that source is most probably correct. If a human encounters evidence for the untrustworthiness of an information source, the trust in that source is decreased. Since Marsh's first attempt in 1994 [99], computer science research has made great efforts in formalizing trust as a computational concept. The *second* goal of this thesis is to re-examine the suitability of trust for an adoption to our problem. To this end, we first analyze to what extent the concept of trust is applicable to automated information acquisition. Motivated by this analysis, we propose a computational trust model and extend the spot-checking mechanism with this model. The trust model can be used to guide the selection of future information sources. Additionally, it helps to reduce the number of challenges that are needed to spot-check trusted sources.

In the remainder of this section, we first restate the overall research problem in a more formal manner. Besides this, we specify the objectives, the methodology, the requirements and the scope of our research. We then show in which concrete distributed systems the research problem actually appears. Afterwards, we summarize the contributions of this thesis. Finally, we provide the reader with the needed terminology and important notations, and describe how the thesis is organized.

1.1 Problem Statement and Objectives

Let an *assertion* (or *piece of information*) be a tuple $(\mathcal{Q}, \mathcal{R})$ that consists of a request $\mathcal{Q}$ and a (corresponding) reply $\mathcal{R}$, where both $\mathcal{Q}$ and $\mathcal{R}$ are formulated in a language $\mathcal{L}$ with well-defined semantics. We call an assertion a *fact*, iff the request $\mathcal{Q}$ is a question and the reply $\mathcal{R}$ contains *all* "correct" answers to this question, and only these – in other words, the reply is *comprehensive* and *valid*. What a correct answer to a question is depends on the semantics and shall not be addressed in this thesis. To give an example, the tuple $(5 + 7 = ?, 12)$ is a fact in a language that interprets "$+$" as regular addition, and 5, 7 and 12 as natural numbers in base ten.

Let $\mathcal{F} \subseteq \mathcal{L} \times \mathcal{L}$ be the set that contains all existing facts in $\mathcal{L}$. Furthermore, let $v(x) : (\mathcal{L} \times \mathcal{L}) \rightarrow \{\text{true}, \text{false}\}$ be a function that states for any assertion whether it is a fact, i.e., whether it is contained in $\mathcal{F}$. We call $v(x)$ an *explicit verification function*. As stated above, an explicit verification function is often too costly to be computed, or is unknown, because for instance it requires background knowledge that an entity does not have.

As shown in Figure 1.1, we look in this thesis at distributed information systems from the perspective of one entity, the "requester", that repeatedly requests information from other entities, the "information providers" or "information sources". To keep things simple, we ignore the requester's role as an information provider, and the information providers' roles as requesters. What is important is that the information providers are autonomous entities and some of them might have malicious intent towards the requester. To match this abstract perspective for instance to a P2P system, both the requester and the information sources would be peers; for a volunteer computing system, the requester would be the

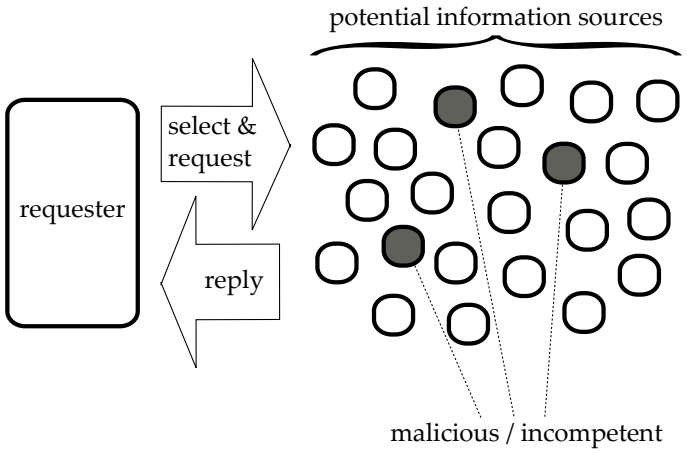


Figure 1.1: Abstract view of a distributed information system.

master, and the information sources would be the workers.

The problem that we address in this thesis can now be stated as follows:

> *Requester A selects an information provider B, makes a request $\mathcal{Q}$ to B, and receives $\mathcal{R}$ as sent by B; given the possibility that B provides incorrect information, how can A ensure that $(\mathcal{Q}, \mathcal{R})$ is a fact, without making use of an explicit verification function $v(x)$?*

1.1.1 Objectives and Methodology

The mechanisms which we propose in this thesis, and that approach the problem as stated above, follow three design objectives:

- Minimize the *overhead*, that is, the ratio of acquired pieces of information to usable pieces of information – for the case of a trustworthy source.
- Maximize the *accuracy* in the estimates of the error rate of acquired information.
- Make the mechanisms *robust* against attack strategies of malicious sources.

Instead of optimizing all three criteria at the same time, the mechanisms that we present require trade-offs to be made. For example, in order to increase accuracy, the overhead has to be increased as well.

For the evaluation of the proposed mechanisms in terms of the above objectives, we use different *methodologies*. When appropriate, we provide a theoretical analysis to show under which circumstances and how far the mechanisms fulfill the objectives. To this end, we either mathematically prove that the claimed properties hold, or provide probabilistic analyses for them. For situations where a theoretical analysis is not adequate, for instance due to complex dynamics, the mechanisms are evaluated experimentally. To this end, we use simulations to measure the accuracy of our mechanisms, and to compare their robustness to that of other mechanisms that have been proposed in the literature. Details on which objectives are evaluated, and how this is done, are given at the respective points in the thesis.

1.1.2 Requirements

Any proposed mechanism has to fulfill two requirements:

- It has to be executable by a computer system without the help of human intervention.
- It must not use an explicit verification function $v(x)$.

The first requirement is needed to make the solutions suitable for fully-automated processes such as is the case in many distributed systems; we make this clearer later in Section 2.4 where we show in which concrete application scenarios our abstract problem occurs. The second requirement makes the mechanisms generic and applicable to different applications. This is because we do not step down to the syntactic/semantic level of the specific language $\mathcal{L}$.

1.1.3 Scope

To keep the scope of the thesis reasonable, we restrict the type of information to which our mechanisms are applicable, as follows:

1. The degree of truth of facts is *binary*, i.e., a reply $\mathcal{R}$ to a request $\mathcal{Q}$ is either correct or incorrect (and nothing in between). In other words, a verification function $v(x) : (\mathcal{L} \times \mathcal{L}) \rightarrow \{\text{true}, \text{false}\}$ for the information exists (see also Sect. 1.1).

2. The acquired information is *objectively* verifiable within the system, i.e., all members of the system would agree on $v(x)$.

Examples of the type of information that fulfills these restrictions are the results of mathematical computations, for instance as they occur in volunteer computing systems. In theory, the result of a computational task is (1) either correct or not, and (2) objective. If in practice the original information has no binary degree of truth, an acceptance range can be defined within which a piece of information is considered to be correct.

As shown in Figure 1.1, the requester can choose from a set of information sources, which we denote by $\mathcal{S}$. Motivated by the volunteer computing scenario, we focus on situations in which all sources in $\mathcal{S}$ are able to reply to a request. In some other application scenarios such as file sharing, only a subset of $\mathcal{S}$ is able to reply to a specific request. Even though we do not explicitly address this case, the mechanisms proposed in this thesis would be applicable. Furthermore, we assume that sources always reply to requests. However, in the outlook of this thesis in Section 7.3.2, we describe an initial idea to deal with non-replying sources.

1.2 Contributions

In this thesis, we make contributions to three approaches to the stated problem, namely in the areas of *spot-checking*, *computational trust* and *replication*. We pay special attention to the possibility of malicious information sources that pursue their goals by using sophisticated strategies. In the following, we outline our main contributions in form of research questions. These questions are further detailed later in the respective chapters.

1.2.1 Spot-Checking with Challenges (Chapter 3)

The idea of spot-checking can be applied in different ways. One way is to spot-check information sources from time to time and to blacklist them if they provide incorrect information too often (e.g., see Sarmenta [133]). Another approach is used in domains where information is allowed to have a certain error rate. There, spot-checking can be used to estimate the error rate of a set of acquired pieces of information. Such an approach has been proposed by Germain-Renaud and Monnier-Ragaigne [60]. For such domains, we suggest a mechanism that addresses the following question:

Q1: *How may we apply spot-checking if the creation of spot checks at runtime is infeasible?*

In this case the number of spot checks, which we call the *challenges*, has to be determined before a request is made, and so the following question arises:

Q2: *What is the optimal number of challenges for a given number of real requests?*

We show how to optimize the number of challenges in respect to the costs caused by (1) the incorrect information that goes undetected and (2) the preparation of the challenges.

Finally, we present a concrete application of spot-checking to an intrusion detection system. As an exception, we are not dealing with binary truth values in this case, but with real-valued estimates about the truth, which itself is still binary; for this scenario, we propose an alternative way of optimizing the number of challenges. In addition to this, we show how a deliberate composition of challenges from different types of challenges can be used to tune the configuration of the system.

1.2.2 Computational Trust in Information Sources (Chapter 4)

In this part of the thesis, we study the computational modeling of trust in information sources. In particular, we address the following questions:

Q3: *What does it mean to "trust in an information source?"*

Q4: *What information can be used as evidence for a source's trustworthiness?*

Having done this, we develop an evidence-based trust model that is based on the spot-checking mechanism proposed in Chapter 3. This trust model provides an answer to the following question:

Q5: *How may we build trust in an information source if we cannot directly verify the information acquired from it?*

In this trust model, we explicitly model the uncertainty that is inherent in a trustworthiness assessment; we prove that it fulfills the two essential properties of uncertainty measures postulated by Wang and Singh [167]. Extending the idea of Quiz [185], we then show how to use trust to reduce the number of challenges. Most evidence-based trust models, and our model in particular, are parameterizable. Thus, the question arises of how to configure these models:

Q6: *How may we systematically choose optimal parameters of evidence-based trust models?*

To approach this question, we propose a tuning procedure for evidence-based trust models that is based on game-theoretic considerations, and apply this procedure to our trust model.

1.2.3 Replication (Chapter 5)

In redundant requesting, the same information is requested redundantly from several information sources. The replies that form a majority/plurality are believed to be correct. This belief is based on the assumptions that information sources act independently and that malicious sources are rare. However, in the scenarios considered here, information providers can communicate among themselves over the Internet without being observed. In addition to this, the Sybil attack enables an attacker to control a considerable number of sources. This makes *collusion attacks* feasible. In this attack, several information sources collectively return the same incorrect information. If they reach the required number of identical replies, their incorrect information is accepted. To counter this main weakness of redundant requesting, our research question is as follows:

Q7: *How may we detect collusion attacks against redundancy with plurality voting?*

To answer this question, we propose a collusion detection algorithm as an alternative to [138]. We evaluate the algorithm with respect to different attacker models, both theoretically and experimentally.

1.3 Organization

The thesis is organized as follows. In Chapter 2, we review existing approaches to the stated problem and point to application scenarios where the problem occurs. The next three chapters contain our main contributions:

- Chapter 3 introduces the probabilistic spot-checking mechanism.
- Chapter 4 presents the computational trust model and shows how to tune it.
- Chapter 5 proposes the collusion detection algorithm.

These chapters also contain the evaluations of the respective mechanisms. In Chapter 6, we put the thesis in the context of related work, and compare our contributions with each other. Finally, we draw conclusions and identify possible future work in Chapter 7.

Chapter 2

Background and State of the Art

In this chapter, we discuss directions that have been taken in various areas of computer science research to address the common problem of ensuring the correctness of acquired information. Following this, we point to concrete application scenarios where this problem occurs.

Organization In Section 2.1, we shortly summarize the historical development of those areas in computer science that deal with the issue of malicious or incompetent information sources. We introduce in Section 2.2 the ideas of fault and sabotage tolerance, and discuss the impact of the "Sybil attack" on these areas. In Section 2.3, we review the main approaches to the considered problem from various fields of computer science research. In Section 2.4, we point to application scenarios where the considered problem occurs, and finally introduce in Section 2.5 terminology and notations that are used in the remaining part of the thesis.

2.1 Historical Development

The emergence of the Internet has facilitated the immediate and automated exchange of information on a global scale. To secure exchanged information, the field of *computer security* has come up with solutions to the problems of authenticity and integrity of data. However, these solutions have not addressed the issue of the creation of incorrect information by malicious or faulty sources. This problem has been investigated in another discipline, distributed systems research, where "fault tolerance" has become a crucial characteristic. For distributed systems, initial solutions were proposed many years ago to tolerate conflicting information coming from faulty components. However, this area did not focus on the intentional and strategic behavior of malicious autonomous components. As the 21st century began, malicious behavior was encountered in concrete volunteer computing and P2P systems. This led to the development of the research field of *sabotage tolerance*, that is tolerance against malicious entities. Apart from this, the idea of using computational trust in the context of computer systems emerged and much research in this direction has been done since.

2.2 Fault and Sabotage Tolerance

Fault tolerance describes the ability of a system to tolerate the failure of some of its components. In distributed systems, fault tolerance is a crucial property with respect to reliability and dependability. To

be robust against data transmission errors, error detection and correction schemes such as *checksums* or *error-correcting codes* can be implemented. To tolerate failures of single computers, multiple machines can redundantly be deployed for the same task, so that the system tolerates some of them failing. This idea of redundancy can also be found in algorithms that address the issue of components that emit conflicting information. Below, we will first look at one fundamental algorithm of this kind. The algorithm addresses the case of failing components, but also of machines that have been compromised by adversaries with the aim of sabotaging the system; Sarmenta has coined the term "sabotage tolerance" [133] to name this kind of tolerance against adversaries. The impact of the Sybil attack on sabotage tolerance is discussed later.

2.2.1 Byzantine Fault Tolerance

In distributed systems, the malfunction of computers can lead to faulty information being sent to other computers. If the computers cannot tolerate the incorrect information, the whole system can destabilize and eventually crash. Lamport et al. [92] reformulate this problem as the one that a group of Byzantine generals faces when be sieging an enemy city. These generals, each of whom has set up camp with his troops somewhere outside the city, can communicate via messenger only. Their aim is to decide at some point in time on a common plan of action, e.g., to attack or to retreat. The messengers are assumed to be reliable and honest. However, there might be traitors among the generals. These traitors represent the malfunctioning or malicious components in a distributed system. A traitor might send different messages to different generals in such a way that eventually some of the generals decide to attack, and others to retreat. The consequence of such a disagreement would be that the whole Byzantine army is defeated. Lamport et al. allow any possible behavior of the traitors. In the literature, this kind of attacker is referred to as a "Byzantine adversary". The Byzantine generals need a procedure that enables them to reach a common decision while tolerating a certain number of traitors. In their paper, Lamport et al. show that *there is no such procedure* if one third or more of the generals are traitors. For the case of $3m + 1$ generals, of whom at most m are traitors, they propose an algorithm that solves the problem. This algorithm assumes that the messages are directly transferred from one general to another, but they do not have to be signed. For the case in which messages can be signed in a way that traitors cannot forge the signatures of loyal generals, the authors propose an algorithm that solves the problem for m traitors and any number of generals. To summarize, these algorithms guarantee that in a distributed system, any conflicting commands that are sent out by a certain number of Byzantine adversaries can be tolerated. This is known as "Byzantine fault tolerance".

2.2.2 The Sybil Attack

The Sybil attack [40] poses a serious threat to "open" systems like P2P or volunteer computing systems. In the Sybil attack, a single attacker generates a great number of fake identities for use in a single system. Without being observed, the attacker can control these identities to orchestrate a collective attack. This attack is possible whenever a fake identity cannot easily be distinguished from a real identity; for instance, in many file sharing systems one person can easily create several identities. Indeed, Douceur [40] has shown that to completely eliminate the attack, (centralized) trusted certification is needed. However, in many of today's systems such an infrastructure is unwanted or not feasible, e.g., in P2P systems. Consequently, alternative approaches have been proposed that try to reduce the effectiveness of the attack [95]. The two basic ideas behind these approaches are:

- to make the creation of an identity costly, or

- to test whether there is a one-to-one relationship between machines and identities, and whether there is a human behind an identity.

To realize the first idea, one can for instance require for each identity to have installed some "trusted device", which itself is costly and so prevents an attacker from creating a great number of identities. However, this also makes it costly for all regular users. To realize the second idea, "resource testing" can be used. In resource testing, one checks whether one identity is backed by the capacity of one machine; however, this was shown by Douceur to be ineffective at least theoretically. Also in practice, if the attacker uses *botnets*[1], resource testing will not help. Therefore, in addition to resource testing, a check of whether a human is behind a machine is required. To this end, "Completely Automated Public Turing tests" (CAPTCHAs) [162] have been proposed. A CAPTCHA challenges an agent with a problem that can be easily solved by a human, but is too hard to be solved by a machine (in a short time); for example, the agent has to type a displayed word that is distorted in such a way that no algorithm could quickly recognize the word, but a human could. To make sure that each identity has a human and not a machine behind it, one can require a CAPTCHA to be solved for each creation. Note that in our context we do not want to challenge an information source with a CAPTCHA each time it provides a piece of information, because we are targeting completely automated systems. Therefore, a CAPTCHA could be requested for each creation of a new identity for an information source. Using this approach, a human could still create thousands of identities in a few hours, because a human can solve a CAPTCHA in a few seconds [31]. Besides, attacks on CAPTCHAs have been found [64, 179, 108]. Thus, it is worthwhile to explore alternative approaches. For instance, could an abnormal increase in the number of identities be seen as evidence for some attacker creating a great number of identities? To detect such a change would require global knowledge about the number of identities, which in general is not easy to obtain in these systems. Also, an attacker could counter this by slowly creating fake identities.[2] Overall, we can conclude that despite a number of different approaches, the possibility of a Sybil attack cannot currently be ruled out.

The Sybil attack can serve as a starting point for launching further attacks. For instance, to attack the protocol from the last section that solves the Byzantine generals problem in the case without signed messages, an attacker could create a number of identities that makes up more than a third of all identities. Also in *reputation systems*, in which entities collectively decide about the trustworthiness of other entities, a large number of identities can be used to distort the real entities' ratings, e.g., by self-promotion of the collaborating entities [71]. As we will discuss in Chapter 6, a trust model too can be affected by a large number of fake identities. Finally, all systems that rely on the assumption that only a small fraction of all sources are malicious, are vulnerable to the Sybil attack, because it provides a way of making this assumption incorrect. This includes for example systems that use majority or plurality voting (e.g., EigenTrust [81]). We will consider another such system in Chapter 5, and propose an algorithm that helps to detect larger fractions of attackers, for instance caused by the Sybil attack.

[1] An attacker can use malicious software to take over private machines over the Internet, and utilize their capacities for his own purposes.

[2] Actually this behavior is also encountered in port scanning, where attackers send only one packet a day, and so go undetected by many intrusion detection systems.

2.3 State of the Art: Ensuring Information Correctness

Several areas in computer science research are confronted with the problem of unreliable information. As a consequence, several approaches have been proposed to solve this problem. In the following we review what we consider to be the main ideas.

2.3.1 Spot-Checking

The idea of *statistical sampling* [107] is to randomly take samples from a population to estimate properties of the whole population. *Spot-checking* uses this technique to "spot-check" a service provider from time to time – without prior warning. In this way, the overall quality of service of a service provider can be assessed. To spot-check an information provider, information can be requested from time to time that is already known or that can be verified easily. If the information provider does not know which information will be checked by the requester, the spot checks can be used to learn about the expected quality of the provided information. In the domain of volunteer computing, several such approaches have been proposed [133, 185, 60]; in Section 6.1, we will explain these approaches in more detail, and compare them to our work. Spot-checking can also be used to check intermediate checkpoints that a source must pass through in order to formulate a final result [41, 127].

To apply statistical sampling to test information sources, certain assumptions have to be fulfilled. If the source is malicious, it will try to identify the spot checks and to respond to these better than to other requests. Therefore, spot checks have to be unidentifiable. This also implies that spot-checking is not done in a predictable way, e.g., every hour, because a source might exploit this knowledge. Besides, a source might be more competent in certain domains, and less competent in others. Thus, depending on the application, sampling has to be made in a *context-sensitive* fashion. Also, the quality of the information provided by a source might change over time. To reflect this, statistical sampling has to be *time-sensitive*, and for instance use forecasting techniques such as trend estimation.

2.3.2 Redundancy

A technique that is currently used in volunteer computing [10] is based on the idea of *redundancy*. We will call this technique "redundant requesting"; in the literature it is called "replication" or simply "redundancy". A piece of information is redundantly requested from several sources that have been picked randomly from a set of potential sources. Only if all, or a majority, or a plurality, of the sources returns the same information, this information is accepted. As long as the probability of a source returning incorrect information is low, the probability that many sources return the same incorrect information is negligible. Redundant requesting is easy to implement and robust against many forms of malicious behavior of sources. Furthermore, it can be used in scenarios where spot checks are too expensive or not possible: In a sense, the sources check each other. However, the mechanism is based on the assumption that the sources are independent. As we will show in Chapter 5, redundant requesting is sensitive to colluding malicious sources, especially if they are numerous – the Sybil attack could be used to this end. Therefore, we propose a mechanism for collusion detection that helps to reduce this threat.

The idea of redundancy can also be used to test global knowledge via some Internet search engine:[3] One can use the count of occurrences that such a search engine displays to vote on alternatives. For example, a person wonders whether it is better to finish an email with "Best wishes" or with "Great wishes". Google returns $25,000,000$ and $17,700$ hits respectively. So, the person might decide on the

[3]For the following experiments, we consulted the Google Inc. search engine on 12-28-2009.

first alternative. In another example, a person wants to know how to spell "neighbor". Google returns around 45 million hits for "neighbor" and around 12 million hits for "neighbour". However, which spelling is correct depends in this case on whether British or American English is intended. This shows that redundancy only works if there is an absolute truth, or one is interested in the "most common truth". Another implicit requirement for redundant requesting is that the requests are truly made in parallel, because the underlying "truth" might change over time. So, in the same way as for spot-checking, redundancy is context- and time-sensitive. Finally, a crucial prerequisite for the applicability of redundancy is that errors in information are generally not positively correlated across the sources. To give an example, a majority of websites might copy incorrect information from the same source website. In this case, redundancy would most probably accept this incorrect information, unless it could trace back the information to its origin; an approach to log the provenance of information is discussed below in Section 2.3.3.3.

2.3.3 Trust and Reputation Systems

A very broad definition of "trust" is given by Diego Gambetta [59]:

> "Trust (or, symmetrically, distrust) is a particular level of the subjective probability with which an agent assesses that another agent or group of agents will perform a particular action, both before he can monitor such action (or independently of his capacity ever to be able to monitor it) and in a context in which it affects his own action [...]."

The mentioned "action" can for instance be the provision of correct information. This definition then reflects the problem that we are facing, namely that the information cannot be checked for its correctness. Thus, "to trust" means that, given that a source is trustworthy, the information it provides can be believed to be correct. The problem is how one can know about the trustworthiness of a source. Below, we will discuss important approaches to assess the trustworthiness of autonomous entities.

Remark: The term "trust" is also used in the context of the "web of trust" (e.g., see [7]). Based on the assumption that trust is transitive in a certain manner, it can be used to build decentralized public key infrastructures (PKI). For instance, if entity A trusts entity B, and B trusts C, then A also trusts C. A PKI can be built along such chains of trust, which in turn helps to ensure data integrity and authenticity. Again, this does not solve the problem of the creation of incorrect information. In our case, we are therefore more concerned with the concept of *evidence-based trust*.

2.3.3.1 Evidence-Based Trust

Evidence-based trust is trust that is derived from past experiences with other entities. The first formal model in this direction was proposed by Marsh [99]. Since then, many other trust models have been proposed (for an overview see for instance [117, 12]). Their common idea is to distinguish between positive and negative experiences, and then to increase or decrease the trust in an entity respectively. How this is done in detail has to be specified by the trust model. For example, a trust model could classify an entity as trustworthy as long as only positive experiences with that entity have been had; it could also compute some numerical trust measure from the gathered evidence. Evidence-based trust can help to increase knowledge about other entities, and in particular about information sources. Simply put, if a trust model classifies an information source as trustworthy, it estimates that the source will provide correct information in the future as well. Because a trust model might incorrectly classify an

untrustworthy source as trustworthy, a certain risk is involved in believing the information provided by a "trustworthy" source. This implies that trust models have to be aware of untrustworthy sources that try to make the model classify them as trustworthy. Generally speaking, a trust model has to be robust against the manipulation strategies of malicious sources. In our case, another problem is caused by our assumption that the acquired information cannot be verified directly; this makes it hard to collect evidence: if one does not know whether the information that a source provided in the past was correct, one has no direct evidence about this source's trustworthiness. In this thesis, we propose a trust model that addresses these two problems.

2.3.3.2 Reputation Systems

In *reputation systems* [125], several entities share information about the trustworthiness of other entities. This information can consist of direct experiences or complete trustworthiness assessments. The information that comes from a reputation system can then be used by single entities to make their trustworthiness assessments of others. The collaboration of entities can speed up the process of detecting malicious sources. Reputation systems have for instance been proposed to fight pollution in file sharing P2P systems [81, 177]. However, a reputation system itself can also be subject to attacks where sources try to promote their trustworthiness by making manipulative reports [71, 157]. Therefore, even for the reputation systems themselves, mechanisms for detecting untrustworthy sources are required.

2.3.3.3 Data Provenance and Data Trustworthiness

Another approach to ensuring the correctness of acquired data is based on combining data provenance with data trustworthiness [33], a rather novel field of research. In data provenance, the path of some data from its origin to its destination is traced, including all intermediate entities, and possible modifications. The data is then presented to a user together with the path(s) it has taken. Based on this, the user can use his knowledge about the trustworthiness of the different entities on the path to decide about the trustworthiness of the data itself. To help the user, a system could provide assistance by displaying knowledge about the trustworthiness of the sources taken for instance from a reputation system. Alternatively, the system could reply to a user's query with the information that appears the most trustworthy.

A combination of data provenance and trust seems to be a promising approach for scenarios where multiple organizations exchange information with each other [19], or for the Semantic Web [39]. It aids in remembering which entity has originally created some piece of information, and which source modified the information, so that afterwards the knowledge about these sources can help in assessing the trustworthiness of the resulting data. However, to this end, the trustworthiness of the sources on the path has to be known. In our thesis, we address the isolated problem of how this trustworthiness can be assessed. Thus, our results are relevant to the approach of combining data provenance with trust. Data provenance on the other hand, is not applicable in many of the scenarios that we are targeting with our thesis. In volunteer computing, for instance, the information passes directly from a worker to the master; thus there is no potentially untrustworthy path that the information takes (the transfer over the Internet can be assumed to be reliable). In other systems, such as classical P2P file sharing systems, only the last information provider matters, which is why this thesis concerns itself with a simple information system as shown in Figure 1.1 (Sect. 1.1).

2.3.4 Plausibility Checking

Sometimes specific knowledge about the information that one wants to acquire can be used to efficiently check the plausibility of the information. Consider the example of a robot that requests the average temperature over the last hour from another robot, which is equipped with a temperature sensor. The requested robot reports a measured value of 400°C. However, the requesting robot knows that the sender's maximal working temperature is at 45°C and that the reported value most probably would have caused it to fail, and making it (highly) implausible. So, a machine can for instance be given intervals outside of which values are implausible. In database systems, this can be done by defining basic semantic constraints that have to be satisfied by the data [116].

A field where plausibility can be used is that of wireless sensor networks. As mentioned in the introduction, the sensors are subject to attacks where for example someone holds a cigarette lighter close to a temperature sensor [165]. Also, the sensors can fail and report incorrect measurements. There have been a variety of approaches to counter this problem [114, 165, 24, 25]. For instance, the causal dependency between measurements of sensors that are close to each other can often be exploited to test the plausibility of the reported measurements. If a temperature sensor measures a very high temperature, and another sensor that is very close measures a much lower temperature, this could be an indicator of the fact that one sensor is reporting incorrect or manipulated information.

Also in the domain of software testing, specific knowledge can be used to efficiently check the results of a computation. Wasserman and Blum [168] proposed a means of checking the results of software at runtime. This becomes feasible through "simple-checkers". A simple-checker tests certain properties of the computed output and recognizes errors with a very high probability. The idea is that the checking is much easier than the computation under test. As an example of a simple checker, they consider the task of sorting an array $\vec{x}$. Sorting requires time $\Omega(n \log n)$, but a specific simple checker can test the resulting array $\vec{y}$ in linear time as follows. First check in linear time whether the elements of $\vec{y}$ are in increasing order. Then check whether $\vec{y}$ is a permutation of $\vec{x}$; having modified the sorting algorithm such that it maintains pointers from the elements in $\vec{x}$ to the elements in $\vec{y}$, this can also be done in linear time. This technique can also be used in cases where a computation is outsourced to another autonomous entity. To reduce the complexity further, spot-checking can be used.

Clearly, the limitation of plausibility checking is that domain-specific knowledge has to be known. Therefore, it is not applicable to general types of information, as we require for our mechanisms.

2.3.5 Symbolic Approaches

In "multi-agent systems" [169, 175], the beliefs of an agent can be represented in form of a propositional belief base, that is as a set of logical propositions that the agent believes to be true. The research field of *belief revision* [58] is concerned with the question of how to integrate a new belief into an existing belief base. However, classical belief revision does not address the issue of the reliability of a source or an observation [52]. In fact, the new belief that has to be integrated is assumed to be correct or at least seen as more reliable than what is already in the belief base. More precisely, to maintain consistency, old beliefs can be dropped, but according to the AGM postulates [9], a new belief will always be part of the revised belief base. Clearly, if the belief that has to be integrated is a statement coming from a malicious or incompetent information source, this approach is not suitable. Another approach would be to accept new beliefs only if they do not conflict with the existing belief set. However, this approach does not address the case where the new belief is actually correct and there is some error in the belief base. In conclusion, as a first step, a proposition should be checked for correctness with some alternative technique, or the

trustworthiness of the source should be taken into consideration. If then the proposition was believed to be correct, it could be integrated into a belief base in a second step. Work in this direction has been done in the area of *non-prioritized belief revision* [69]. How the correctness of some piece of information, and in particular a proposition, can be assessed, is addressed in our thesis.

2.3.6 Incentive Compatibility

Instead of recognizing incorrect information, one can try to prevent its creation in the first place. This concerns the case of malicious information sources that intentionally provide incorrect information. If a malicious source is rational, it will do so only if it can expect to get some utility out of it. *Incentive-compatible mechanisms* try to reduce this utility by giving incentives to act honestly. The precondition for this is that the utility function of the malicious sources is known, and that it is generally possible to provide incentives which will have an impact on this utility function. In some domains like electronic commerce or commercial P2P computing where monetary incentives can be provided, such mechanisms are promising (e.g. [80, 65, 181]). However, these mechanisms are not applicable if a malicious source has its highest utility if the utility of the other entities in the system is minimized; imagine a source that provides incorrect information just because it enjoys sabotaging the system. Also, in volunteer computing or file sharing systems, it is not clear what suitable incentives could be.

2.3.7 Encrypted Computation and Code Obfuscation

As mentioned in the introduction, malicious workers that return incorrect results pose a serious threat to the reliability of (volunteer) distributed computing systems. As a consequence, several mechanisms have been proposed in the literature that address the problem of result checking. A closer look at this field of research is taken in the section on related work (Sect. 6.1), where we compare the proposed mechanisms to ours. At this point, we mention approaches that take a slightly different view of the problem of incorrect results than ours. The aim of these approaches is to make it infeasible for a malicious host to modify the result computation code in a purposeful way. Two of these approaches are *encrypted computation* and *obfuscation* (for a more detailed description see for instance Appendix B in [132]). In encrypted computation, the requester encrypts the original function of interest f with some encryption scheme E to give $E(f)$. The encrypted function is then transferred to a worker, who is expected to apply it to some input x. After the computation, the worker returns the result, which is $E(f(x))$, to the requester. The requester finally applies the corresponding decryption function E^{-1} to the result, and finds $f(x)$. Consider the following illustrative though naive example, where k denotes an integer constant:

$$f(x) = e^x \,, \tag{2.1}$$

$$E_k(f(x)) = f(x) + k \,, \tag{2.2}$$

$$E_k^{-1}(g(x)) = g(x) - k \,. \tag{2.3}$$

Here the encryption function simply adds some integer to the original function. For decrypting the result, the same integer has to be subtracted. To provide more privacy, the requester can use additional techniques such as homomorphic encryption that allow the input also to be processed in encrypted form.

The idea of obfuscation is to transform the original program code into equivalent code that is very hard to understand. Many different code transformations can be applied; to give a very simple example,

irrelevant code can be added or bogus data dependencies can be introduced. It is important that these transformations make it hard for humans and also for algorithms to "deobfuscate" the code.

Both encrypted computation and obfuscation make it hard for an attacker to modify the code in a reasonable way. However, they do not prevent a malicious host from computing the correct results, but still returning incorrect results, e.g., random results. To this end, checksums and watermarking techniques can be inserted into the code such that certain patterns appear in a correct result. The checksums or watermarks have to be easy to check. While encrypted computation generates a large overhead and is currently limited to specific computations, obfuscation cannot easily been proven to be secure. According to Sarmenta [132], a composition of different techniques from this area might provide a final solution. Since these techniques are limited to result checking, we consider more generic approaches in our work instead.

2.3.8 Inherent Fault and Sabotage Tolerance

There is evidence that specific types of parallel algorithms have the property of inherently tolerating a certain number of incorrect and/or dropped results. For the latter, Gonzales et al. [67] provided first evidence that, in the context of volunteer computing, this is actually the case for the use of *genetic algorithms*. A genetic algorithm is a stochastic optimization technique that is inspired by the process of natural selection (see also [73, 63]). It starts with an initial population of individuals whose parameters can be set randomly. By means of *selection*, *reproduction* and *mutation* the population evolves over several generations, and optimizes itself with respect to a fixed *fitness function*. The algorithm usually stops if either the individuals cease to improve, or a maximum number of generations is reached. The fact that the search is performed with several individuals reduces the probability of getting trapped in local maxima [63]; but it also lets several parallel search threads interact with each other [129, p. 116]. A genetic algorithm is well-suited for parallel execution, because each computation of the fitness function, which is usually the most costly part, can be run on a separate machine; the only point where synchronization has to take place is when all individuals of one generation have finished their fitness computation. It is apparently the case that a certain number of dropped results can be tolerated. It can also be conjectured that errors could be tolerated, because in a subsequent generation these errors are likely to be canceled out. However, the robustness of genetic algorithms in the face of orchestrated attacks, and of large fractions of malicious workers, seems to be an open issue in research. A collective of workers could for instance try to direct the genetic algorithm towards a bogus maximum of the fitness function.

2.4 Problem Scenarios

In many distributed information systems, autonomous entities create or modify information that is later accessed and used by other entities in the system. How these distributed information systems are configured, and how the abstract problem that we have defined in the previous section appears there, is detailed in the following by giving illustrative example scenarios. For many of these scenarios, we point to the incentives that entities have to intentionally provide incorrect information.

Volunteer Computing A volunteer computing system [132] is a centralized distributed computing system to perform large-scale parallel computations. As already mentioned in the first part of the introduction, voluntarily participating people offer the resources of their private PCs, the "workers", usually in form of idle CPU cycles. Compared to the high acquisition and maintenance costs of centralized su-

percomputers, volunteer computing is very low-priced. This is one of the reasons why the concept has become very successful and numerous projects have been launched (e.g., [3, 1, 15, 132, 11, 2, 4]). However, the drawback of using private PCs is that they might return incorrect results because they fail due to overclocking or software errors [105, 156], or because of malicious intent [10], e.g., because the workers want to harm the master, or because they want to save resources (e.g., by returning random results). One of the incentives for volunteers to cheat in SETI@home has been to improve their position in the ranking that is published by the system and that shows how much a volunteer has contributed [105]. Removing this ranking is not a solution since it would also affect the incentives of many volunteers to participate. Alternatively, for specific computations, "simple-checkers" can be used to verify results in an efficient manner [168]. Unfortunately, no such mechanism is known for general computations [65]. This shows the need for efficient and generic mechanisms that check results for correctness. Although this thesis approaches the problem from a more abstract perspective, the evaluations of our mechanisms are applicable to the conditions in the scenario of volunteer computing.

File Sharing *Peer-to-peer (P2P) systems* can be defined as distributed computer systems where autonomous nodes ("peers") share distributed computer resources such as content, storage or computing power [103]. *File sharing* is a well-known class of P2P systems that enables its peers to exchange digitally-stored information, such as audio or video files. Nowadays, it accounts for the largest part of Internet traffic [183]. In [134], Saroiu et al. analyze two popular file sharing systems. One of their findings is that "peers tend to deliberately misreport information if there is an incentive to do so". They suggest that future systems need mechanisms to directly measure and verify reported information. Certain types of information can directly be verified. For instance, the reported bandwidth of a peer can be checked against its actual download speed. However, if information cannot be directly verified, mechanisms are needed that check information correctness in alternative ways. Examples for this kind of information are incorrectly labeled music files, which cannot easily be uncovered by machines. This fact is exploited by the *pollution attack* [96] where damaged files are inserted into the system and then spread among the peers. The reason for this attack is that the music industry wants to distract people from illegally downloading copyrighted material; here, the music industry has monetary incentives to carry out this attack. The authors of [96] showed that in KaZaa more than 50% of the copies of many popular files were actually "polluted". There are two ways in which a file can be polluted; either the content itself is damaged (*content pollution*) or the description of the file does not match the content (*metadata pollution*). The second attack especially cannot be easily discovered by a computer without human help, because a computer does not in general know what, for example, a file named "J. S. Bach - Johannes Passion" should sound like.

Wireless Ad Hoc Networks A *Wireless Ad Hoc Network* [158] is a network consisting of autonomous wireless nodes that collaboratively organize routing infrastructure in an ad hoc manner. The efficiency and fairness of these networks in terms of throughput depends heavily on the correctness of exchanged routing information. There is an incentive to manipulate this information, because nodes are limited in battery power [139] and so try to minimize their own battery consumption. Thus, a node A could for instance misreport to its neighbor B that a certain node C is not reachable from A, in order to cause B to use an alternative route.

Wireless Sensor Networks A *Wireless Sensor Network* (WSN) [8] is a network that consists of wireless nodes that are equipped with sensors. The nodes are spatially distributed so that the sensors cover

a certain area in which they can measure environmental conditions such as temperature, humidity or pressure [8]. The measurement data is transferred over multiple hops to a base station. In these networks, the sensors might be subject to attacks. Wagner [165] gives the example of a network consisting of sensors that measure the temperature in different rooms of a building; the average temperature of all sensors is used to regulate the central air conditioning unit. Now, assume that there is a person that wants the temperature in his room to decrease. This person can attack the system by holding a lighted match close to a sensor, which will cause the sensor to report an erroneous room temperature; as a consequence, and in line with the attacker's goal, the average measured temperature increases and the air conditioning unit will decrease the temperature in all rooms. This shows the need for checking information reported by the sensors.

Web services A *Web service* is a "software system designed to support interoperable machine-to-machine interaction over a network" [5]. To give an example, a Web service could offer the multiplication of large matrices. In general, information that is provided by a Web service can be incorrect. As in the example of matrix multiplication, the requesting machine does not want to explicitly verify the results. To this end, mechanisms are required that help the requester machine to efficiently check the information.

From a more general point of view, a Web service can be seen as a consultant that advises a client. For this scenario, we showed in [143] that there are actually situations where consultants have incentives to be ill-informed, and thus to provide information which is not sufficiently checked for correctness. How strong these incentives are depends among other things on the fraction of ill-informed consultants among all consultants, and the importance that a client attaches to the consultant's reputation and price respectively.

Public Knowledge Stores Wikipedia [6] is a well-known system that enables humans to share public knowledge over the World Wide Web. Because no semantics are annotated, machines are not able to process the information on a semantic level. However, the proposed Semantic Web [18, 136] makes information on the web processable by machines. Also, a provision of such information by means of distributed hash tables (e.g., [118, 153, 128, 184]) is imaginable. The peculiarity of public knowledge systems in this vision is that no trusted entity could guarantee the correctness of the available knowledge. However, without such a guarantee, autonomous data processing could not rely on information coming from these systems. This would especially be the case in the presence of malicious participants that try to manipulate the systems. Humans would be necessary to manually check the correctness of the information. For large systems however this would not be feasible. Thus, when making public knowledge systems usable for automated data processing, one key issue will be to provide mechanisms that can check whether the provided knowledge is correct.

2.5 Terminology and Notations

2.5.1 Basic Classification of Information Sources

In the main part of the thesis, we consider a basic classification of information sources by two basic characteristics, their *attitude* towards the requester and their *competence*. The attitude of an information provider can be *honest* or *malicious*. Honest information providers exclusively provide information that they believe to be comprehensive and valid with respect to the request; to be more formal, if the request is $\mathcal{Q}$, then they always provide $\mathcal{R}$, while believing that $(\mathcal{Q}, \mathcal{R})$ is a fact. Malicious information providers

Table 2.1: Basic classification of information sources.

Attitude	Competence	Class
honest	high	*trustworthy*
honest	low	*unreliable*
malicious	high	*untrustworthy*
malicious	low	-

Table 2.2: Confusion matrix

		actual class	
		+	-
classification	+	*true positive*	*false positive*
	-	*false negative*	*true negative*

in contrast provide as well information they believe to be incorrect or incomplete. The competence of an information provider can be *high* or *low*. By competence we refer to the ability to make sure that the provided information is comprehensive and valid. In other words, a competent information provider is more often correct in his beliefs about facts. Table 2.1 shows the four resulting classes of information providers. Clearly, the ideal information source is a *trustworthy* information provider. *Unreliable* or *untrustworthy* information providers should be avoided. The difference between these two types is that an unreliable information provider is not aware of his incompetence and so unintentionally provides incorrect information, whereas an untrustworthy information provider intentionally provides incorrect information but at the same time tries to stay undetected. As a consequence an untrustworthy information provider is in general harder to identify. Finally, there is a fourth class of information providers, namely those that are both malicious and incompetent. Since they are not able to appear competent they should not be able to successfully pretend to be trustworthy. Therefore, the main focus of our work is on the first three classes of information sources. In Section 4.1.3, we take a closer look at the attitudes and competences of information providers.

2.5.2 Notations

For probabilities, we use Jaynes' notations [77]: capital letters for random variables, and small letters for numerical values of the respective random variables. For example, X denotes a random variable, and $p(x)$ is short for $p(X = x)$. Analogously, we write capital P to denote discrete probability distributions, e.g., $P(X)$; we use lowercase p for single probabilities, e.g., $p(x)$. For continuous distributions, we use lower case function symbols, e.g., $f(x)$.

To denote a set of values $\{x_1, \ldots, x_k\}$, we use the notation $\{x_i\}_1^k$ or short $\{x_i\}_i$; analogously we write $(x_i)_i$ for tuples. To label the outputs of binary classification, we use the common terminology illustrated in form of a "confusion matrix" in Table 2.2. The symbols that are used in this thesis are itemized and explained in the nomenclature below the appendix on page 131.

Chapter 3

Spot-Checking with Challenges

In this chapter, we first present a probabilistic mechanism for assessing the error rate of acquired information without making use of explicit verification. This mechanism mixes requests with "challenges", which are requests for already-known facts. In the second part of the chapter, we apply the concept of challenges to an intrusion detection system. Challenges allow for an evaluation and adaptation of the system at runtime, and also for an adjustment of the system to counter more harmful threats.

Organization In Section 3.1, we describe the core spot-checking mechanism. Because challenges can be costly or difficult to formulate, it is important to keep the number of challenges used small. For this reason, we show in Section 3.2 how to choose an optimal number of challenges. In Section 3.3, the mechanism is analyzed both in terms of its robustness against attackers and its efficiency. Finally, we show in Section 3.4 how challenges can be used to test the possibly unreliable components of an intrusion detection system.

This chapter is related to [150, 151] and [121, 122, 123].

3.1 Mechanism

The following mechanism is suggested by the way humans "spot-check" articles in newspapers or magazines. As well as searching for flaws in arguments made by the article, a reader checks the credibility of an article given the knowledge he already has. The more contradictions with his knowledge that a reader finds, the less reliable he thinks the article is. Our mechanism uses this idea in that it randomly merges the requests for the information of interest with some "challenges", i.e., requests for already-known facts. The replies to the challenges can be verified, and so the correctness of the remaining information can be estimated. Provided that challenges cannot be identified, a malicious information source can only decide on the number of incorrect answers that it is going to return. The more incorrect answers it returns, the more it runs the risk to be uncovered.

In the following, we assume that an information provider has already been selected, and that it is going to reply to all requests. The manner of selecting an information source is addressed later in Chapter 4.

3.1.1 Procedure

Let $\mathcal{F}$ again be the set of all facts, and let $\mathcal{K} \subset \mathcal{F}$ be the set of facts known by the requester, i.e., its knowledge. Let M be the set of "real requests", i.e., requests for which the correct replies are not yet known. The requester picks a set of requests N for which the correct replies are known; formally this means:

$$\forall \mathcal{Q} \in N : \exists \mathcal{R} \in \mathcal{L} : (\mathcal{Q}, \mathcal{R}) \in \mathcal{K} \, . \tag{3.1}$$

We call the requests in N "challenges". It is important that the information source cannot distinguish the challenges from the real requests. The requester randomly merges M and N and sends the resulting set of requests $M \cup N$ to an information source. The information source then returns $|M \cup N|$ replies. The replies to the challenges can directly be verified: by checking whether the challenges together with the corresponding replies are facts. As a result, one finds y incorrect replies to the challenges. Based on this number, one can estimate the number of incorrect replies to the real requests M (as shown below). If the probability of an acceptable number of incorrect replies to the real requests is high enough, the replies can be used. Otherwise they are discarded.

3.1.2 Correctness Estimates

At the outset, we need to make two assumptions: The correctness of different pieces of information is statistically independent; and there is no way for an attacker to distinguish the challenges from the real requests.

Let Z be the random variable of the overall number of incorrect replies to the whole request $M \cup N$. Let Y be the random variable of the incorrect replies to the challenges only. We want to know $P(Z|Y)$, because $Z - Y$ is the number of incorrect replies to the real requests M, and the value of Y is known. Let X denote this random variable. *Bayes' theorem*[17] states:

$$P(Z|Y) = \frac{P(Y|Z)P(Z)}{P(Y)} \, . \tag{3.2}$$

We can compute the denominator by marginalization over all possible values of Z. This gives for a concrete value of Y:

$$p(y) = \sum_z p(z)p(y|z) \, . \tag{3.3}$$

Thus, we need only to show how to compute $P(Y|Z)$ and $P(Z)$. Because the challenges are indistinguishable from the real requests, and the correctness of the replies is statistically independent (basic assumption), the incorrect replies are randomly distributed among M and N. Therefore, we can solve $P(Y|Z)$ with basic combinatorial considerations. There are $\binom{|N|}{y}$ possibilities for having y incorrect replies to the challenges. Having z incorrect replies overall, the real requests get $z - y$ incorrect replies. So, for a given z, there are, for a certain y, $\binom{|N|}{y}\binom{|M|}{z-y}$ many equally likely cases. These are called the "favorable cases". This allows to compute the probability of a certain y for a given z by dividing the number of favorable cases, those for the specific y, by all possible cases to distribute z incorrect replies

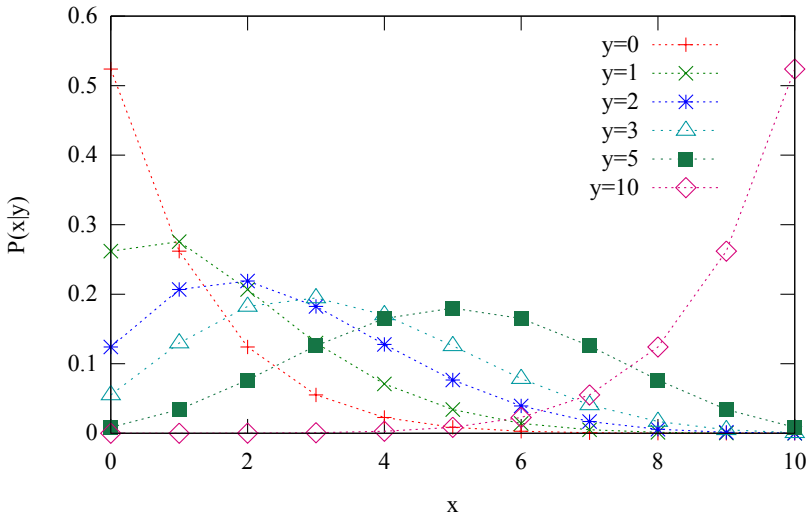


Figure 3.1: Probability distributions for incorrect replies ($m = n = 10$, uniform prior, PMF).

among $M \cup N$:

$$p(y|z) = \frac{\binom{|M|}{z-y}\binom{|N|}{y}}{\binom{|M \cup N|}{z}} . \tag{3.4}$$

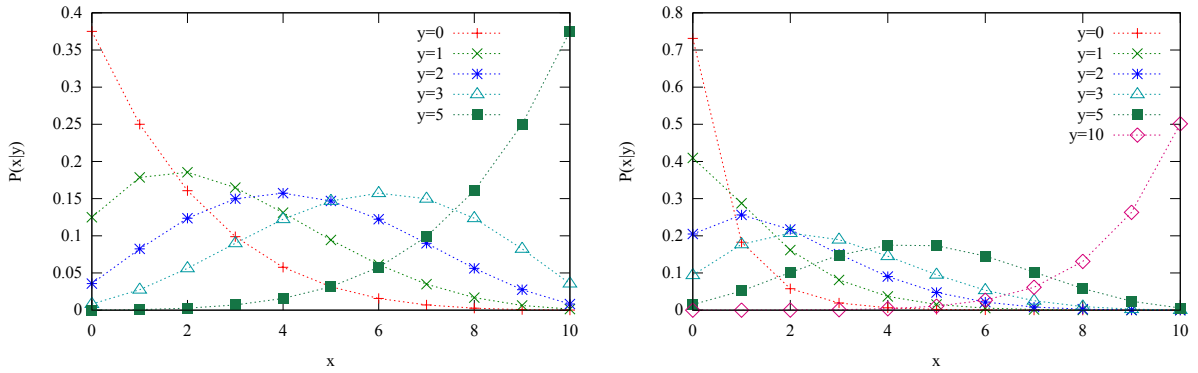
For the "prior probability distribution" $P(Z)$, or simply the "prior", we have to make an assumption on how probable a certain number k of incorrect replies is, with $k \in \{0, \ldots, |M \cup N|\}$. In the general case, a uniform prior can be assumed, i.e., $p(z) = 1/(|M \cup N| + 1)$. This assumption is based on the *principle of indifference* [77, p. 40]; the principle states that if n mutually exclusive and exhaustive events are indistinguishable, except for their names, then they must be assigned probability $1/n$ each. However, domain knowledge can be used to define a more accurate prior. For instance, if it is known that information providers return correct information most of the time, $P(Z)$ can be defined lower for higher z values; of course, it still has to hold that $\sum_z p(z) = 1$. A simple example for a non-uniform prior would be the following:

$$p(z) \propto \frac{1}{z+1} . \tag{3.5}$$

Example 3.1 (A uniform prior)**.** *Let X denote the number of incorrect replies to the real requests, i.e., $X = Z - Y$. Then, $P(X|Y)$ simply equals $P(Z - Y|Y)$. As eq.* (3.4) *indicates, the probabilities of the values of X depend not only on the number of incorrect answers to the challenges, but also on how many real requests ($|M|$) and challenges ($|N|$) have been used. We write m for $|M|$ and n for $|N|$. We first show examples for $m = n = 10$.*

Figure 3.1 shows the probability mass functions *(PMF) of different y for a uniform prior; for instance, if* 3 *incorrect replies to the challenges are found ($y = 3$), the probability for also having* 3 *incorrect replies to the real requests is around* 0.2*.*

Example 3.2 (Changing n)**.** *Figure 3.2(a) shows the PMF for the case when five challenges are used to estimate the correctness of ten real requests. Since fewer challenges are used, the probability distributions are now less determined; for instance compare the function for $y = 2$ with that from Figure 3.1. We show later how this*



(a) $m = 10$, $n = 5$, uniform prior, PMF

(b) $m = n = 10$, skewed prior, PMF

Figure 3.2: Probability distributions for incorrect replies.

characteristic can be expressed in the form of the "risk".

Example 3.3 (Changing the prior)**.** *For other priors, the shapes of the distributions change. To give an example, Figure 3.2(b) shows the PMF for a prior as in eq.* (3.5)*. Because high numbers of incorrect replies are less probable, the probability distributions are skewed to the left; this becomes evident when comparing $n = 0$ and $n = 10$, which are not symmetric with each other.*

Example 3.4 (A binomial prior)**.** *A special case occurs when the prior is chosen to be a* binomial distribution *with parameter p:*

$$p(z) = \binom{m+n}{z} p^z (1-p)^{m+n+z} \,. \tag{3.6}$$

This distribution describes the probability of z successes in $m + n$ independent Bernoulli trials with success probability p each. Accordingly, we get m independent Bernoulli trials to determine X, and n to determine Y. As a result, the posterior distributions of X and Y are also binomial distributions with parameter p, independent of each other.[1] *In other words, the knowledge of y does not tell us anything about X; and this holds independently of the number of challenges n. However, as we show in the next chapter, the knowledge of y is also helpful in this case – it can be used to estimate the error rate of the binomial distribution.*

3.1.3 Accept Decision

Probability distributions can be drawn in form of a *cumulative distribution function (CDF)*. Such a CDF is shown in Figure 3.3 for the same distribution as in Figure 3.1. The function $\mathrm{CDF}(x)$ is defined as the sum of the PMF up to x, i.e., in our case it is $\mathrm{CDF}_y(x) = \sum_{x' \le x} p(x'|y)$. The CDF shows how probable an upper bound of incorrect replies to M is; for instance, if no incorrect replies to the challenges are found ($y = 0$), there are with a probability around 0.91, not more than 2 errors among the real requests. Furthermore, the CDF demonstrates how many incorrect replies to the challenges a requester is willing to tolerate. For example, assume the requester wants to accept M only if at most 3 incorrect replies can be expected with probability greater than 0.8. Then he accepts only those y whose CDF falls inside the gray "area of acceptance" in Figure 3.3; in our example this is $y = 0$ and $y = 1$. In general, let $i_{\max}$

[1] We show this formally in Appendix A.1.

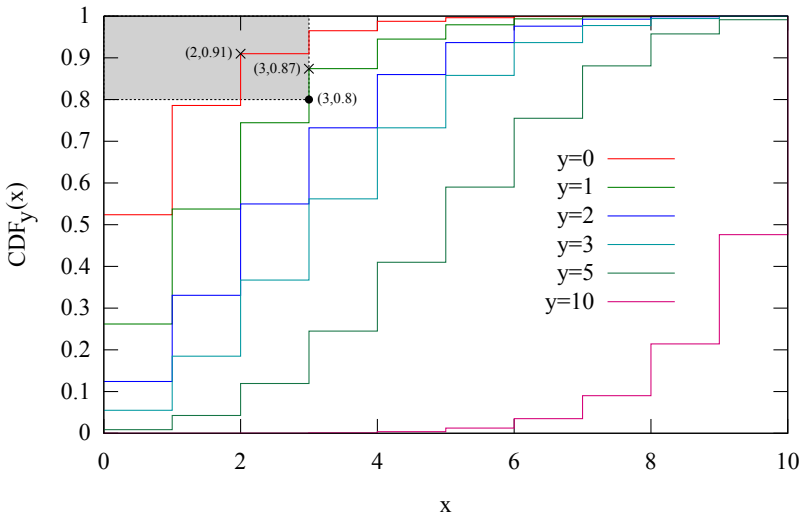


Figure 3.3: Area of acceptance ($m = n = 10$, uniform prior, CDF).

denote the maximal number of incorrect replies to M that are tolerated for a probability of at least $p_{\min}$. Then, the requester accepts a set of replies only if the following holds:

$$\mathrm{CDF}_y(i_{\max}) \geq p_{\min} \ . \tag{3.7}$$

Note that also the expected value of the probability distribution of Z might be helpful for making certain decisions. If a certain number y of incorrect replies to the challenges is found, the expected value of incorrect replies overall (Z) can be computed based on eq. (3.2):

$$\mathbb{E}(Z|y) = \sum_{z_i} z_i \cdot p(z_i|y) \ . \tag{3.8}$$

In conjunction with the corresponding standard deviation, the expected value could alternatively be used to make the accept decision. However, in our case, we only need a guarantee on the upper bound of the number of incorrect replies, for which the CDF is more suitable.

3.1.4 Creation of Challenges

Under the assumption that challenges are indistinguishable from real requests, the spot-checking mechanism gives probabilistic guarantees about the error rate of the unverified replies. Since we take an abstract perspective in this thesis, we have not proposed how such challenges may be created. In the scenario of volunteer computing, the challenges can be precomputed on trusted hosts as proposed in [185], or consist of simple checkers (see Sect. 2.3.4). In general, if an entity knows a set of known facts $\{(\mathcal{Q}_i, \mathcal{R}_i)\}_i$, it can use the respective requests $\mathcal{Q}_i$ as challenges. This is safe from a privacy point of view, because the actual replies $\mathcal{R}_i$ are kept secret. If the challenges are accepted responses to real requests, it is not completely certain that they are correct. In this case, the probability computations described in the previous section would have to be adapted to take this uncertainty into account. Similarly, as we discuss later in Section 6.2, challenges can be a product of the application of redundant requesting. Again, one has to account for the possibility of faulty challenges.

3.2 Optimal Number of Challenges

In this section, we show how to find an optimal number of challenges with respect to: the *costs* of a challenge, and the *risk* of the estimate of z. We first show how the risk of an estimate of z can be determined. The risk describes how incorrect the estimate is expected to be. In our case, the risk depends mainly on the number of challenges n. Simply put, the more challenges are used, the better the estimate can be expected to be.

Let z' denote the actual number of incorrect replies in a concrete application of the procedure; m real requests and n challenges are used. If a certain number of y incorrect replies to the challenges has been found, the estimate $P(Z|y)$ can be computed as shown in eq. (3.2). The error E of this estimate can be measured as follows:[2]

$$E(m,n,y,z') = \sum_{z_i=0}^{m+n} p(z_i|y) \cdot |z_i - z'| \ . \tag{3.9}$$

The domain of this error measure is $[0,m]$. Each z' occurs according to the prior with some probability $p(z')$, and so we can compute the risk of an estimate, that is, the expected error in the estimate. For n challenges and m real the risk can be computed as follows:

$$R(m,n) = \sum_{z'} p(z') \sum_{y} p(y|z') \cdot E(m,n,y,z') \ . \tag{3.10}$$

Let the following cost parameters be given:

- c_c = costs for one challenge,
- c_r = costs for taking a *risk* of $R(m,n) = 1$.

In practice, the costs for one challenge have to reflect the costs for creating the challenge, and also things like the price that has to be paid to an information provider for a reply to that challenge. Based on these costs, the overall cost function can be defined:

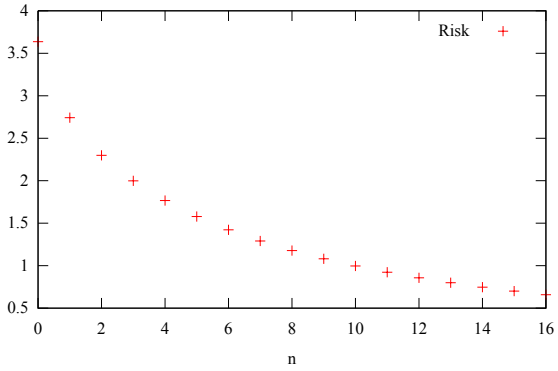
$$\mathrm{C}(m,n,c_c,c_r) = n \cdot c_c + R(m,n) \cdot c_r \ . \tag{3.11}$$

To find the optimal number of challenges n_{opt} for given m, c_c and c_r, the n has to be chosen for which $\mathrm{C}(m,n,c_c,c_r)$ is minimal:

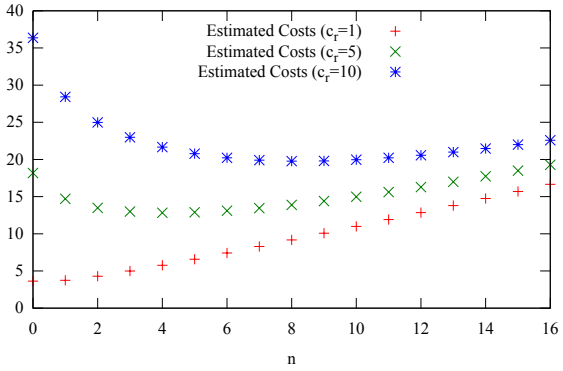
$$n_{\mathrm{opt}} = \operatorname*{argmin}_n \mathrm{C}(m,n,c_c,c_r) \ . \tag{3.12}$$

Example 3.5 (Optimal n)**.** *For now, we consider the case of a uniform distribution for Z, and use a uniform prior for the estimate. Figure 3.4(a) shows the risk for different numbers of challenges n, when the number of real requests is $m = 10$. For $n = 0$, this shows how great the risk of the estimate is if no challenge is used; in this case, the estimated probability distribution is the prior, i.e., the (discrete) uniform distribution. Figure 3.4(b) illustrates the resulting cost functions $\mathrm{C}(m,n,c_c,c_r)$ for different cost parameters. The exact costs are shown in Table 3.1. In this table, the boxes mark the minima, and so the respective n_{opt} can be found in the corresponding row.*

[2] Instead of the absolute value $|z_i - z'|$, the square $(z_i - z')^2$ can be used, as is done for the "mean standard error" (MSE). The square has the property of more heavily penalizing incorrect estimates. Whether this is desired or not is not discussed at this point.



(a) $R(m, n)$ for $m = 10$.



(b) $C(m, n, c_c, c_r)$ for $m = 10$ and $c_c = 1$.

Figure 3.4: Risk and costs.

Table 3.1: Estimated costs $C(m, n, c_c, c_r)$ for $m = 10$ and $c_c = 1$.

n	$c_r = 1$	$c_r = 5$	$c_r = 10$
0	3.6363636	18.181818	36.363636
1	3.7424242	14.712121	28.424242
2	4.2985028	13.492514	24.985028
3	4.9989571	12.994785	22.989571
4	5.7666248	12.833124	21.666248
5	6.5783745	12.891872	20.783745
6	7.4221946	13.110973	20.221946
7	8.2905012	13.452506	19.905012
8	9.1780495	13.890248	19.780495
9	10.08103	14.405149	19.810298
10	10.996584	14.98292	19.96584
⋮	⋮	⋮	⋮

3.3 Analysis

We analyze the mechanism in two ways: first, in terms of *robustness* against attacks, and secondly in terms of *efficiency*. The *accuracy* and *certainty* of an estimate have already been addressed in Section 3.2.

3.3.1 Security Analysis

In this section, we want first to analyze how an attacker could decide on an optimal number of incorrect replies, which enables us to argue which prior is the most robust. Secondly, we discuss the possibility of "intersection attacks" and propose a countermeasure. Finally, we look into the issue of checking a source's identity.

3.3.1.1 A Robust Prior

In the following analysis, we assign the attacker the ability to answer each request correctly, i.e., we assume that the attacker is what we call "competent" (see Sect. 2.5.1). This allows the attacker to explicitly choose a certain number of incorrect replies. If an attacker does not have this ability, his attacks can only be less effective. In addition to this, we assume a strong attacker that knows how many challenges are used. If n were completely unknown to the attacker, he could not reason about the optimal number of incorrect replies.

In the accept decision, the requester, or the "defender", wants to have at most $i_{\max}$ incorrect replies to M with probability at least $p_{\min}$. The requester will accept a reply having y incorrect replies to the challenges only if eq. (3.7) holds. Otherwise, the requester will reject the reply. Let us say that in the case of rejection, an attack is not successful. Thus, an attacker tries to maximize the probability of his replies being accepted. At the same time, the attacker tries to maximize the effectiveness of his attack, i.e., the number of incorrect replies. To make a decision, the attacker needs to know the probability that an attack with z incorrect replies will be successful. The probability of getting z incorrect replies accepted depends on $i_{\max}$ and $p_{\min}$, and on the prior used. As defined above, the two numbers $i_{\max}$ and $p_{\min}$ determine for how many incorrect replies to the challenges (y) the overall reply is still accepted – in other words, how "tolerant" the requester is. Let A_Y denote the set of tolerated ys; for example, if $A_Y = \{0, 1\}$, then the requester accepts the overall reply if he finds zero or one incorrect replies to the challenges. The probability of an attack with z incorrect replies being successful can then be computed as follows:

$$p(\text{accepted}|z) = \sum_{y \in A_Y} p(y|z) \; . \tag{3.13}$$

We can add up the probabilities for different ys, because they are mutually exclusive events. How to compute $P(Y|Z)$ is shown above in eq. (3.4).

Example 3.6 (Success probabilities)**.** *Consider the case of $i_{max} = 2$, $p_{min} = 0.9$ and $m = n = 10$. Then a request is already discarded if* 1 *incorrect reply to the challenges is found, as can be seen in Figure 3.3 ($CDF(2) = 0.9097744$ for $n = 0$). Therefore, $A_Y = \{0\}$, and the attacker gets the following probabilities:*

$$P(\textit{accepted}|Z) = P(Y = 0|Z) \; . \tag{3.14}$$

Table 3.2 shows the resulting values. If, for instance, the attacker uses only one incorrect reply, the attack is successful with probability 0.5*. This is because the probability of this incorrect reply falling on a challenge is* $10/20 = 0.5$*. Note that $P(accepted|Z)$ is not normalized, i.e., $\sum_z p(accepted|z) \neq 1$.*

We can describe an attack strategy as a probability distribution $P(Z)$. It is not evident what attack strategy a rational attacker would probably choose. This is because we do not know what the attacker's preference is: to maximize the success probability ($p(\text{accepted}|z)$) *or* to maximize the number of incorrect replies (z). In the first case, the attacker would tend to return fewer incorrect replies, because then the success probability is higher; in the second case, he would tend to return more incorrect replies. Now, if we use a prior skewed to the right, we will overestimate the actual error rate in case of the first type of attacker; if we use a prior skewed to the left, we will underestimate the error rate in the second case. Hence, we argue that a uniform prior should be used because it accounts equally for the different attack strategies. One could argue that a prior skewed to the right would motivate an attacker to return fewer incorrect replies, because fewer errors would lead to a higher error rate estimate; but especially

Table 3.2: Probabilities of successful attacks ($m = n = 10$).

z	$p(\text{accepted}\|z)$
0	1
1	0.5
2	0.2368421
3	0.1052632
4	0.0433437
5	0.0162539
6	0.0054180
7	0.0015480
8	0.0003572
9	0.0000595
10	0.0000054
⋮	⋮
20	0

when dealing with larger numbers of trustworthy sources with small error rates, which should be the usual case, one would very often make large estimation errors.

3.3.1.2 Intersection Attacks

For the repeated application of the proposed mechanism, the choice of the challenges has to be made reasonably. An imprudent choice of challenges can give attackers the possibility of intersecting request sets and so identifying challenges. This violates our assumption of indistinguishable challenges: Attackers could selectively make correct and incorrect replies, which would cause the requester's estimate of Z to be wrong. To give an example, consider an attacker that replies correctly to a whole set of requests $M \cup N$. The attacker can be sure that the requester will accept the replies. If one of the requests in $M \cup N$ appears again later, either to the same information provider or to another one who colludes, the provider can identify the request as a challenge. We call attacks of this kind "intersection attacks". In our case, we can make these attacks impossible. Consider two cases:

Case 1: A set of requests $M \cup N$ is rejected, because y is too high. Consequently, the requester wants to request M again.

Case 2: A set of real requests M is requested for the first time.

In the first case, the requester should use the same challenges N again. In the second case, the requester should use challenges that have never been used before. It is easy to see that these two rules ensure that for any two sets of requests R_1 and R_2 the following holds:

$$(R_1 \cap R_2 = \emptyset) \vee (R_1 = R_2)\ . \tag{3.15}$$

In this way, attackers cannot learn anything about the challenges from intersecting different sets of requests.

3.3.1.3 Identity

One big advantage of the mechanism is clearly that it can also be used if the identity of an information source cannot be checked. As long as the correctness of the replies is statistically independent, the mechanism can be applied to any set of information. On the other hand, the mechanism is necessarily limited in the sense that it does not learn about information providers. In Chapter 4, we therefore equip the mechanism with learning functionality.

3.3.2 Efficiency

We analyze the efficiency of the proposed mechanism in terms of computational complexity and overhead caused by the challenges.

3.3.2.1 Computational Complexity

The computational complexity of the spot-checking mechanism is in $\mathcal{O}(m+n)$, which is determined by the computation of eq. (3.3). The decision as to whether to accept a set of replies is only one comparison (eq. (3.7)). The complexity can be reduced in practice: every estimate can be precomputed for different m, n and y, which only leaves the challenges to be checked at runtime ($\mathcal{O}(n)$). This is an upper bound, because the comparisons can be stopped as soon as it is clear that $i_{\max}$ cannot be fulfilled with probability $p_{\min}$ (as similarly proposed in [60]).

3.3.2.2 Challenge Overhead

With spot-checking, the requester can adjust the overhead of verification continuously. If the proposed mechanism is used in isolation, its *overhead* can be measured as the ratio of all requests to the number of results that matter:

$$\text{Overhead}(\text{spot-checking}) = \frac{m+n}{m} \,. \tag{3.16}$$

In the redundant requesting approach, which we discuss later in Chapter 5, the requester has to choose a positive integer $k > 1$ as redundancy level (usually $k = 2$ or $k = 3$ [11]). To give our spot-checking mechanism the same overhead as redundancy with $k = 2$, the same number of challenges and real requests can be used ($n/m = 1$). Nevertheless, it has to be kept in mind that spot-checking entails a risk of misestimation.

3.4 Runtime Evaluation and Adaptation of an Intrusion Detection System

In this section, we show how spot-checking with challenges can be applied to a concrete system for *network intrusion detection*. In contrast to the basic spot-checking with challenges introduced above, we will now use challenges to particularly to assess the accuracy of an information source – and not the error rate of the replies. In the remaining part of this chapter, the information sources are assumed to be under our control and do not try to fool our mechanism; in fact, they do "the best they can". In the next chapter, where we deal with assessing the trustworthiness of a source, we focus on the possibility of malicious sources that use strategies in order to go undetected. Thus, for now, we are not dealing with the *trustworthiness* of a possibly malicious source, but with the *accuracy* of a honest source.

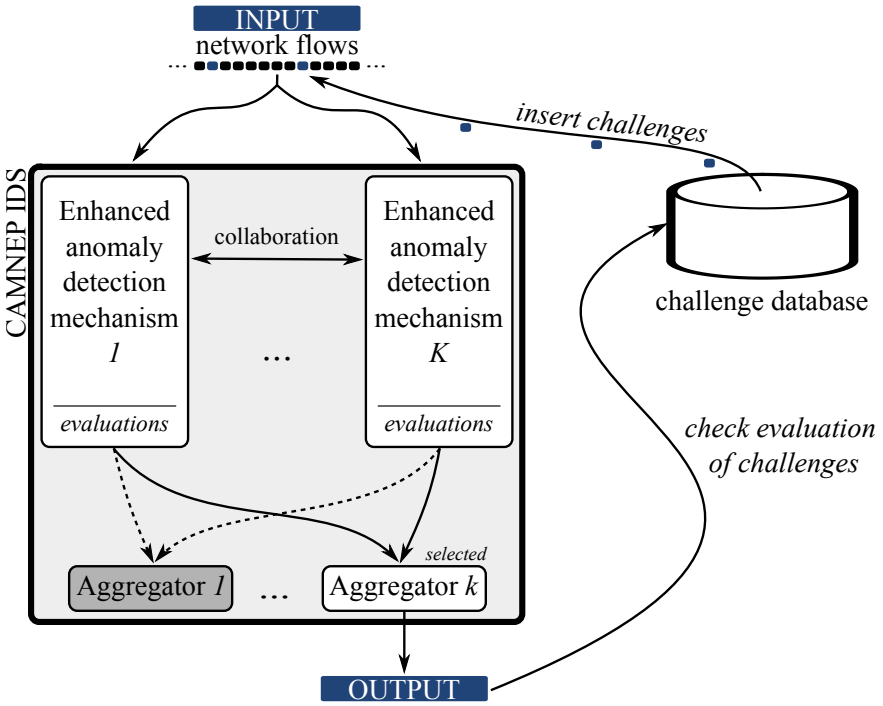


Figure 3.5: The CAMNEP IDS with runtime evaluation.

Intrusion Detection with CAMNEP The input to a network intrusion detection system (IDS) is information about the packets entering the network under surveillance. The aim of the system is to report packets that provide evidence for an ongoing attack on the network. The IDS that we consider here is the *CAMNEP* IDS [120] developed at the Czech Technical University in Prague. CAMNEP gets as input *network flows*, which represent groups of packets using the same protocol (TCP/UDP/ICMP) and the same source and destination (determined by IP addresses and ports). The flows contain also information such as the number of packets, bytes, etc. For each flow, the system outputs a value in $[0,1]$ that estimates the likelihood of the flow being part of an attack; we call these values *attack scores*. The closer the attack score is to 1, the more legitimate the flow is thought to be; 0 represents the most suspect flows. To present the results to a network administrator, thresholds are applied to decide when to raise an alarm.

Figure 3.5 illustrates the CAMNEP system together with the extension for evaluation and adaptation at runtime. The core CAMNEP system consists of a set of collaborating components, each of which deploys a different algorithm for *anomaly detection*. Anomaly detection aims to learn what observations are "normal", and then to report abnormal observations. This idea can be used in network scenarios where legitimate flows make up the great majority of the traffic, and so attacks often belong to the abnormal traffic. In the literature, many different approaches to anomaly detection have been proposed for network intrusion detection [93]. The aim of combining several anomaly detection approaches is to outweigh the weaknesses[3] of the different approaches. This is also motivated by the success of ensemble techniques for machine learning, such as *bagging* [21] and *boosting* [50]; in these techniques several instances of a classifier are trained on different subsets of the training set, and then the classifications of the different instances are aggregated. The output of the different anomaly components of CAMNEP

[3]Lazarevic et al. [93] have found that the anomaly detection schemes considered are especially suitable to detect certain attacks.

are aggregated by the k "aggregators" in different ways (see [120] for details). One of the aggregators is selected and provides the final output of the system, namely the attack scores for the current flows. To use the terms defined in this thesis' introduction: The aggregators act as information sources, the network flows are requests ($\mathcal{Q}$), and the attack scores are the replies.[4]

Selecting Aggregators The CAMNEP system is continually learning, and so the accuracy of the different aggregators is very likely to change over time. Also, some aggregators will be more accurate concerning certain attacks than others. Therefore, it has to be decided at runtime what the best aggregator currently is. To this end, we propose to add prepared challenges to the regular network flows (see Figure 3.5). These challenges, which are stored in a database, are flows that are known to be either attack flows or legitimate flows. Each aggregator can then be evaluated by checking how well it processed the challenge flows; eventually, the most accurate aggregator is selected. A big advantage of the selection of different aggregators at runtime is that it is hard for attackers to predict the behavior of the IDS. The benefit of using challenges is that by a deliberate choice of challenges, the system can be directed to attach more importance to specific attacks, as we show later in Section 3.4.3; before this, we show how the aggregators are evaluated (Sect. 3.4.1), and then how a reasonable number of challenges can be determined (Sect. 3.4.2).

Intrusion Detection Networks The procedure proposed in the following addresses the case where the anomaly detection mechanisms are run in a trusted environment. This is commonly the case, which means that we are dealing with trustworthy but possibly unreliable information sources. However, in "intrusion detection networks" (IDN), several IDS are distributed over the Internet and act as collaborating peers and exchange information about intrusions. In this case, peers can be untrustworthy information sources. Our approach can be applied in this case: Malicious challenges can be sent to other peers in order to test whether these peers actually raise an alarm. However, since an alarm is only raised for malicious challenges, the peers can be rated only on these. For information on related research, the reader is referred to [76, 42, 56, 57].

3.4.1 Evaluation of Aggregators

The challenge database contains flows that belong to known attacks (we call them *malicious*), and flows that are known to be legitimate. Challenges from both types are added to the input of CAMNEP. After the challenges have been processed, each aggregator can be evaluated. The performance of an aggregator for the malicious and the legitimate challenges is measured separately. Let $\alpha_1, \ldots, \alpha_n$ be the attack scores that a certain aggregator assigns to the *malicious challenges*. The sample mean and sample standard deviation of the attack scores for malicious challenges are then:

$$\bar{\alpha} = \frac{1}{n} \sum_i \alpha_i \,, \tag{3.17}$$

$$s_\alpha = \sqrt{\frac{1}{n-1} \sum_i (\alpha_i - \bar{\alpha})^2} \,. \tag{3.18}$$

[4] Only when a threshold is applied to an attack score, one gets a response $\mathcal{R}$, for which it can be checked whether $(\mathcal{Q}, \mathcal{R})$ is a fact. However, because information is lost by applying a threshold, we consider the real-valued attack scores in the following.

Analogously, let $\lambda_1, \ldots, \lambda_1$ be the attack scores that the same aggregator assigns to the *legitimate challenges*; sample mean and sample standard deviation are:

$$\bar{\lambda} = \frac{1}{n} \sum_i \lambda_i \,, \tag{3.19}$$

$$s_\lambda = \sqrt{\frac{1}{n-1} \sum_i (\lambda_i - \bar{\lambda})^2} \,. \tag{3.20}$$

The sample means $\bar{\alpha}$ and $\bar{\lambda}$ are *unbiased estimators* of the respective real means μ_α and μ_λ of the underlying distributions (see [107, p. 278]); this means that their expected values equal the real means ($\mathbb{E}(\bar{\alpha}) = \mu_\alpha$ and $\mathbb{E}(\bar{\lambda}) = \mu_\lambda$). To make the sample standard deviations unbiased estimators as well, they are computed with $\frac{1}{n-1}$ instead of $\frac{1}{n}$ (see [141, p. 158]).

As mentioned before, an attack score close to 0 marks attacks; a score close to 1 marks legitimate traffic. So, for a very accurate aggregator, $\bar{\alpha}$ is close to 0, $\bar{\lambda}$ is close to 1 and s_α and s_λ are small. Rewarding a high accuracy, we give an aggregator the following intermediate rating (higher values are better):

$$r_i = \frac{\bar{\lambda} - \bar{\alpha}}{s_\lambda + s_\alpha} \,. \tag{3.21}$$

We continuously evaluate aggregators in batches of five minutes each:[5] index i of r_i counts the batches, and the highest i marks the most recent batch. Based on a number of batches, we compute the overall rating for an aggregator as follows:

$$r = \sum_i w_i \cdot r_i \,, \tag{3.22}$$

where w_i are weights that determine how much older ratings should be considered; the weights are normalized so that it holds: $\sum_i w_i = 1$. In the current implementation, we use weights that decrease exponentially and discard all ratings older than 5 batches. Finally, the aggregator with the highest current rating r provides the system output for the time of the next batch.

3.4.2 Adaptive Number of Challenges

In this section we show what a reasonable number of challenges is. Since attack scores are real-valued, we have to take a different approach from that of Section 3.2.

We must meet two requirements in determining the number of challenges. First, enough challenges should be used to make the evaluation of the aggregators meaningful. Secondly, as few challenges as possible should be used because the challenges have an impact on the learning inside of CAMNEP; e.g., if too many challenges mimicking a certain attack are used, the anomaly detection mechanisms will start to consider these flows as normal traffic. To account for this, we impose an upper bound $n_{\max}$ on the number of challenges.

3.4.2.1 Applying Margins of Error

We can give guarantees about the accuracy of the aggregator ratings by using "margins of error". A margin of error is a bound that is not violated with a certain *confidence level* C, which is simply a probability. In Appendix B, we show how to compute the margin of error for means and standard deviations.

[5] This is a common time interval used in network flow processing.

When applying margins of error, we have to satisfy two preconditions: First, the challenges have to be randomly picked from the challenge database, and secondly, the challenge database has to be representative of the whole diversity of network flows.

In the following we write Δ_L for a left margin of error for a mean (or a standard deviation); a margin of error Δ_L computed for an estimated mean $\bar{\mu}$ states that the actual mean μ is greater than $\bar{\mu} - \Delta_L$ with probability $\mathcal{C}$. In the same way, there is a right margin of error Δ_R; for a mean this states that the actual mean μ is guaranteed to be smaller than $\mu + \Delta_R$. We use Δ_L and Δ_R to denote margins of error for both means and standard deviations; the context makes clear which is meant.

Recall that from the latest batch we know for each aggregator the two means $\bar{\alpha}$ and $\bar{\lambda}$, and the respective standard deviations s_α and s_λ; these are based on an equal number of malicious (α) and legitimate (λ) challenges. Given this, we can order the aggregators (a) based on their most recent ratings resulting from eq. (3.21):

$$a_1 \geq a_2 \geq \cdots \geq a_k \; . \tag{3.23}$$

We can make sure that the ratings from the next batch do not reverse this ordering by chance. To this end, we choose for every pair of aggregators (a_i, a_{i+1}) a number of challenges that guarantees a margin of error that ensures the order. Let us look at aggregators a_1 and a_2, where a_1 has currently the better rating. Then for the latest batch i it holds (indices 1 and 2 identify the two aggregators):

$$r_{i,1} = \frac{\bar{\lambda}_1 - \bar{\alpha}_1}{s_{\lambda_1} + s_{\alpha_1}} \geq \frac{\bar{\lambda}_2 - \bar{\alpha}_2}{s_{\lambda_2} + s_{\alpha_2}} = r_{i,2} \; . \tag{3.24}$$

Considering these estimates as the best predictions for the performance on the next batch, we can apply the error margins to each estimate as follows:

$$\underbrace{\frac{\overbrace{(\bar{\lambda}_1 - \Delta_L) - (\bar{\alpha}_1 + \Delta_R)}^{\text{lower bound}}}{(s_{\lambda_1} + \Delta_R) + (s_{\alpha_1} + \Delta_R)}}_{\text{upper bound}} \geq \underbrace{\frac{\overbrace{(\bar{\lambda}_2 + \Delta_R) - (\bar{\alpha}_2 - \Delta_L)}^{\text{upper bound}}}{(s_{\lambda_2} - \Delta_L) + (s_{\alpha_2} - \Delta_L)}}_{\text{lower bound}} \; . \tag{3.25}$$

The margins of error are chosen in such a way that, for an increasing number of challenges n, the lower bounds are strictly increasing, and the upper bounds are strictly decreasing. Because the error margins converge to zero for $n \to \infty$, there must be an n for which the inequality holds, unless we have $r_{i,1} = r_{i,2}$. In the latter case we choose $n_{\max}$. Otherwise we start with a number of challenges $n = 2$, compute this formula for increasing n, and choose the first n for which the inequality holds; if this n is larger than $n_{\max}$, we choose $n_{\max}$ instead.

We compute such a number of challenges for every pair of consecutive aggregators; we write n_i for the number of challenges determined for the pair of aggregators (a_i, a_{i+1}). Because we are testing each aggregator with the challenges inserted to the whole system, we select as overall number of challenges:

$$2 \cdot \max_{i<k}(n_i) \; . \tag{3.26}$$

The factor 2 accounts for the fact that we need n malicious and n legitimate flows.

3.4.2.2 Computational Complexity

We showed in Section B that each margin of error can be computed in $\mathcal{O}(1)$, and so the computation of eq. (3.25) for one n is also in $\mathcal{O}(1)$. Additionally, to gain performance, we apply the error margins in eq. (3.25) separately, instead of computing overall error margins for the two terms (on the left and right side). In the latter case, we would have to solve $2\sum_{i=1}^{k-1} n_i$ integrals of complex probability distributions at runtime, as shown in Section B. However, the overall margins of error would be narrower, and so our approach sacrifices some precision for performance. The drawback of the loss of precision is that we probably overestimate the required number of challenges. On the other hand, this way we get an even higher confidence that the inequality will not be violated by chance.

In eq. (3.26), we considered all aggregators. To reduce complexity for the determination of n, one can bound the number of considered aggregators to the first $k_{\max} \leq k$ aggregators, so that only $k_{\max}$ many n_is have to be computed. The rationale for the restriction of k is that for the system output, it only matters which aggregator is at the first position, and aggregators with very poor ratings do not compete with the most accurate ones. For the next batch, every aggregator is then evaluated based on the following number of challenges:

$$2 \cdot \max_{i<k_{\max}} (n_i) \; . \tag{3.27}$$

For each pair of consecutive aggregators, eq. (3.25) has to be computed up to $n_{\max}$ times, which gives an overall complexity of $\mathcal{O}(k_{\max} \cdot n_{\max})$.

3.4.3 Threat-Model Driven Challenge Selection

The challenges that are used to test the aggregators in CAMNEP are network flows that either are malicious, i.e., belong to an attack, or are part of legitimate traffic. In the previous section, we assumed that both malicious and legitimate challenges are picked randomly from the challenge database. Alternatively, we can select the malicious challenges in a way so that they better represent more harmful attacks. As a result, the aggregator that is chosen performs best on these selected challenges, and so the IDS is directed at detecting the harmful attacks in particular.

First of all, we group attack flows into t different attack classes $\mathtt{A}_1, \ldots, \mathtt{A}_t$; one class contains *horizontal scans*, while another class contains *buffer overflow* attempts, etc. Attacks from the same attack class share certain characteristics, and so an anomaly detection mechanism that is able to detect one attack from a class will be likely to detect other attacks from the same class. Also, the expected damage of attacks belonging to the same class is similar; let $D(\mathtt{A}_i)$ denote the expected damage of an attack from class $\mathtt{A}_i$. Our aim is to make the detection of more damaging attacks CAMNEP's highest priority. So, if for instance an attack from $\mathtt{A}_1$ is twice as damaging as an attack from $\mathtt{A}_2$, then it is also twice as important to detect an attack from $\mathtt{A}_1$. To account for this, we use a number of malicious challenges that is *proportional to the expected damage* of the class they belong to. As a consequence, we get a more accurate assessment of an aggregator's performance for the damaging attacks, and give this accuracy a higher weight in the aggregator's rating in eq. (3.21).

It remains to show how to determine the damage of an attack class. To this end, we make use of "attack trees"; an attack tree structures the ways in which attackers can realize a specific threat. After assigning expected damages to concrete threats, we can derive the damage of an attack class. In the following, we first explain in more detail what an attack tree is. We will then show how the representation of an attack tree can be minimized. Finally, we show how for a given set of attack trees, and associated

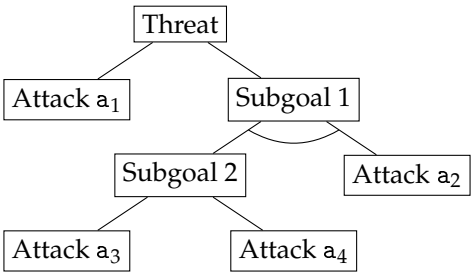


Figure 3.6: Example attack tree.

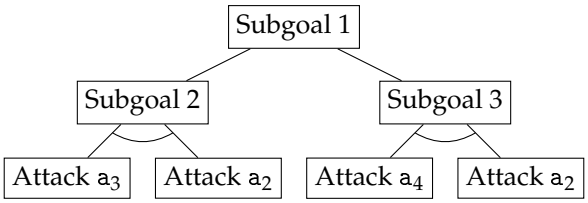


Figure 3.7: Subtree.

expected damages, the damage of the different attack classes can be determined.

3.4.3.1 Attack Trees

There are often several alternative ways in which an attacker can realize an attack. An *attack tree* structures these different ways. As described in [106], an attack tree is composed of AND and OR branches. Figure 3.6 illustrates the structure of such an attack tree. In this figure, the branch with an additional arc depicts an *AND branch*; all other branches are *OR branches*. The root constitutes the *threat* that an attacker can try to implement, for instance a "server takeover". To reach the root, an attacker needs to carry out a number of attacks represented by the leafs of the tree, and so moves from the bottom to the top. In the case of AND branches, an attacker has to reach all children of a node to reach the node. In the case of OR branches, it suffices to reach one of the children of a node to reach the node. For example, in attack tree in Figure 3.6, an attacker can reach the subgoal 2 by carrying out attack a_3, then can reach subgoal 1, by additionally carrying out attack a_2, and by reaching subgoal 1 has accomplished the threat.

Minimal Representation of Attack Trees The attack tree in Figure 3.6 could be drawn in different ways, each describing the same set of possible attacks; for example, "Subgoal 1" could be replaced by the subtree shown in Figure 3.7. In fact, the interpretation of AND and OR branches as *boolean operators* makes clear how equivalent attack trees can be constructed. Based on the boolean interpretation, we show in the following how to find the minimal and unique representation of an attack tree. To this end an attack tree is translated into a propositional logic formula, which is minimized with traditional techniques.

An attack tree T can be transformed into a propositional logic formula as follows. Initially, the formula consists only of the root of the tree. We now successively go through the tree, from the top to the bottom. At each step, we replace nodes by their children, connecting them with the appropriate logic operation ($\vee$ for OR branches, $\wedge$ for AND branches). Parentheses are used to group the children

together. When there is no node left that can be replaced, we are done. For the example tree shown in Figure 3.6 this process is as follows:

$$1.) \quad \text{“threat”} , \tag{3.28}$$

$$2.) \quad (\mathtt{a}_1) \vee (\text{“subgoal 1”}) , \tag{3.29}$$

$$3.) \quad (\mathtt{a}_1) \vee ((\text{“subgoal 2”}) \wedge (\mathtt{a}_2)) , \tag{3.30}$$

$$4.) \quad (\mathtt{a}_1) \vee (((\mathtt{a}_3) \vee (\mathtt{a}_4)) \wedge (\mathtt{a}_2)) . \tag{3.31}$$

A formula is in *Disjunctive Normal Form* (DNF) iff it is a disjunction of conjunctive clauses. A formula is called “canonical” if all clauses contain all variables. We can bring any formula into canonical DNF by building a truth table that contains all variables, and using all rows that evaluate to *true* as the clauses. For our running example that results in:

$$(\mathtt{a}_1 \wedge \mathtt{a}_2 \wedge \mathtt{a}_3 \wedge \mathtt{a}_4) \tag{3.32}$$

$$\vee(\mathtt{a}_1 \wedge \mathtt{a}_2 \wedge \mathtt{a}_3 \wedge \neg\mathtt{a}_4) \tag{3.33}$$

$$\vee(\mathtt{a}_1 \wedge \mathtt{a}_2 \wedge \neg\mathtt{a}_3 \wedge \mathtt{a}_4) \tag{3.34}$$

$$\vee \qquad \ldots \tag{3.35}$$

However, there is much redundancy in the formula. For example, lines 3.32 and 3.33 together are logically equivalent to $\mathtt{a}_1 \wedge \mathtt{a}_2 \wedge \mathtt{a}_3$. To minimize the formula and remove all redundancy, we apply the Quine-McCluskey algorithm [115]. For the attack tree in Figure 3.6, we get:

$$(\mathtt{a}_1) \vee (\mathtt{a}_3 \wedge \mathtt{a}_2) \vee (\mathtt{a}_4 \wedge \mathtt{a}_2) . \tag{3.36}$$

Having an attack tree in such a minimal DNF, we can say that an attacker realizes the threat if he manages to make at least one clause true.

A formula in DNF can be represented as a set of clauses $\{C_1, C_2, \ldots\}$ where each clause C_i is a finite set of attacks. We write $F(T)$ for the minimal DNF formula in set notation that corresponds to attack tree T. For example, attack tree from Figure 3.6 is then:

$$F(T) = \{\{\mathtt{a}_1\}, \{\mathtt{a}_2, \mathtt{a}_3\}, \{\mathtt{a}_2, \mathtt{a}_4\}\} . \tag{3.37}$$

Alternative Minimization Technique A different technique for minimizing attack trees can be found in [100]; in their approach, the authors define transformations directly on attack trees, whereas our approach works with transformations on propositional logic formulae. Their approach is not equivalent to ours. In their approach, the tree corresponding to $(\mathtt{a}_1) \vee (\mathtt{a}_1 \wedge \mathtt{a}_2)$ is for instance not further simplified (they do not use “absorption laws”), while in our approach it is minimized to $\mathtt{a}_1$. As a consequence, the procedure that we propose below in Section 3.4.3.2, would in many cases yield different results for an attack tree minimized with their and our method. Thus, their approach cannot be used here.

Attack Graphs A more general approach to model threats is “attack graphs” (see [137]). In this approach, a threat is represented by a graph, analogously to attack trees. Attack graphs that do not contain cycles can be represented in form of attack trees (we allow each attack to appear several times in a tree). Thus, in these cases, our procedure of threat-model driven challenge selection works also for attack graphs. If an attack graph contains cycles that cannot be removed, the attack graph representation is

more powerful and cannot be captured by an attack tree. For this case, [102, 135] propose alternative techniques that use the PageRank algorithm [22] to measure the threat posed by single attacks.

3.4.3.2 Expected Damage for Attack Classes

Let a set of attack trees $\mathcal{T} = \{T_1, \ldots, T_n\}$ be given. In addition to this, the associated expected damages $D(T_1), \ldots, D(T_n)$ have to be defined manually. We now want to know what the overall expected damage for an attack class $\mathtt{A}_i$ is. In other words, we want to transfer the damage that has been associated with an attack tree, and so is also associated with the attacks in this tree, via these attacks to the attack classes.

In a first step, we determine how important an attack $\mathtt{a}_i$ is in realizing a threat represented by attack tree T_j. Consider the following points:

- For making a chosen clause true, an attacker needs to make true *all* attacks in this clause. Thus, all attacks belonging to the same clause are equally important.

- An attacker realizes a threat corresponding to T_i if he satisfies at least one clause in formula $F(T_i)$. Hence, any satisfied clause in $F(T_i)$ causes damage $D(T_i)$, and so all clauses within a formula are equally important.

Since we are dealing with minimized formulas, these two requirements are easy to meet – we can compute the "importance" of an attack $\mathtt{a}_i$ within a tree T_j as follows:

$$I(\mathtt{a}_i, T_j) \stackrel{\text{def}}{=} \frac{1}{|F(T_j)|} \sum_{\substack{C_k \in F(T_j), \\ \text{with } \mathtt{a}_i \in C_k}} \frac{1}{|C_k|} \,, \tag{3.38}$$

where $|C_k|$ is the number of attacks in clause C_k, and $|F(T_j)|$ is the number of clauses in the formula T_j. The reader can easily verify that, if attack $\mathtt{a}_i$ is not in T_j, then its importance within the tree is zero; also, the sum of the importances of all attacks in the tree is 1 (see App. A.2 for a proof).

In a second step, we want to know which damage can be attributed to a single attack. The higher the expected damage of an attack tree is, the higher is the expected damage of any single attack in the tree. So, we can weight the importance of an attack $\mathtt{a}_i$ within a tree T_j with the damage of the respective threat $D(T_j)$. We get the estimated damage for a single attack $\mathtt{a}_i$ by summing over all attack trees:

$$D(\mathtt{a}_i) \stackrel{\text{def}}{=} \sum_{T_k \in \mathcal{T}} D(T_k) \cdot I(\mathtt{a}_i, T_k) \,. \tag{3.39}$$

Because the importances of all attacks within a tree add up to 1 (see above), the sum of the damages of all attacks equals the overall damage of all threats (see App. A.2 for a proof). This property ensures that we do not add or leave out any damage at any point in our process.

Finally, we can compute the damage of a whole attack class $\mathtt{A}_i$ by adding the damage of all attacks in this class:

$$D(\mathtt{A}_i) \stackrel{\text{def}}{=} \sum_{\mathtt{a}_j \in \mathtt{A}_i} D(\mathtt{a}_j) \,. \tag{3.40}$$

Let n be the overall number of malicious challenges used. We then use a number of malicious challenges

from each attack class $\mathtt{A}_i$ that is proportional to its damage $D(\mathtt{A}_i)$:

$$n_{\mathtt{A}_i} = \frac{D(\mathtt{A}_i)}{\sum_j D(\mathtt{A}_j)} \cdot n \ . \tag{3.41}$$

In Appendix C, an example illustrates the whole process of deriving the damage of attack classes.

3.4.3.3 Applicability of Error Margins

Note that by selectively picking challenges, challenges are no longer randomly picked from the whole set of challenges, but only within each attack class. However, in the previous section, we made the assumption of randomly-picked challenges: Only if we globally select challenges randomly, can we apply the margins of error as described in Section 3.4.2.1. One approach to address this issue is to compute an aggregator rating r for each attack class separately. As a consequence, a new ordering between the aggregators has to be defined that accounts for the different ratings; the easiest approach would be to compute a weighted average of all ratings, where the weights are the expected damages of the respective attack classes.

Alternatively, one can pick the challenges randomly from the whole database, but compute the mean attack score for the malicious flows $\bar{\alpha}$ as a weighted average [98, p. 250]:

$$\bar{\alpha}' = \frac{\sum_i D(\mathtt{A}(\alpha_i)) \cdot \alpha_i}{\sum_j D(\mathtt{A}_j)} \ , \tag{3.42}$$

where $\mathtt{A}(\alpha_i)$ denotes the attack class to which attack α_i belongs. The weighted standard deviation can be computed as follows [98, p. 251]:

$$s'_\alpha = \sqrt{\frac{\sum_i D(\mathtt{A}(\alpha_i)) \cdot (\alpha_i - \bar{\alpha}')^2}{1 - \sum_j D(\mathtt{A}(\alpha_j))}} \ . \tag{3.43}$$

The aggregator rating in eq. (3.21) can be then computed with $\bar{\alpha}'$ and s'_α. However, for the computation of the number of challenges in Section 3.4.2, the common mean and standard deviations have to be used. This allows us to combine the approach for computing a good number of challenges with the approach for applying higher weight to important attack classes. The disadvantage is that there is the possibility, because the challenges are picked randomly from the whole set, that very few or no challenges are selected from the most important attack classes, and thus the accuracy in these cases would drop.

3.4.4 Evaluation

In this section we describe experiments that tested the performance of CAMNEP with the approaches for adapting the number and the composition of challenges at runtime. The experiments have been conducted by members of the Agent Technology Center at the Czech Technical University, Prague, under the lead of Martin Rehák. For the experiments, five different anomaly detection mechanisms [178, 91, 90, 44, 142] were used. The resulting outputs were aggregated by 30 different aggregators using *weighted* or *order-weighted* averaging, or a combination of both. In weighted averaging, the outputs of the different anomaly components are weighted more or less strongly. In order-weighted averaging the values are weighted in respect to their ordering, e.g., the first and last values in the ordering are weighted more heavily than values in the middle.

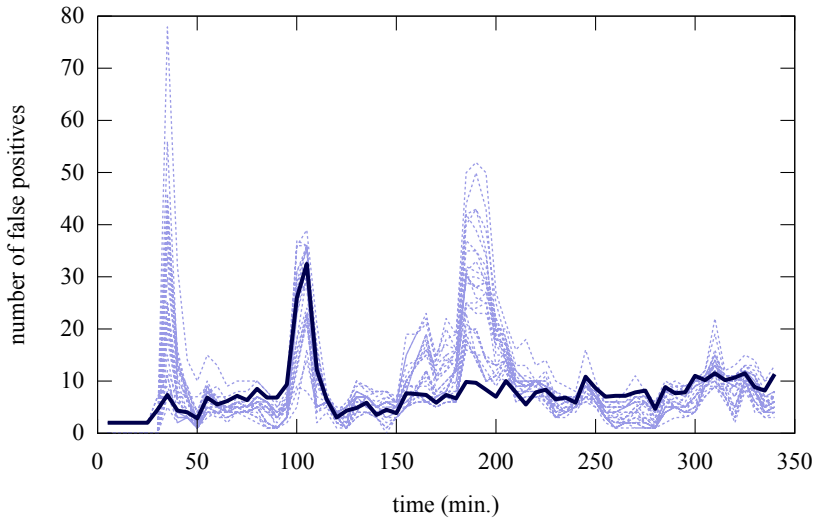
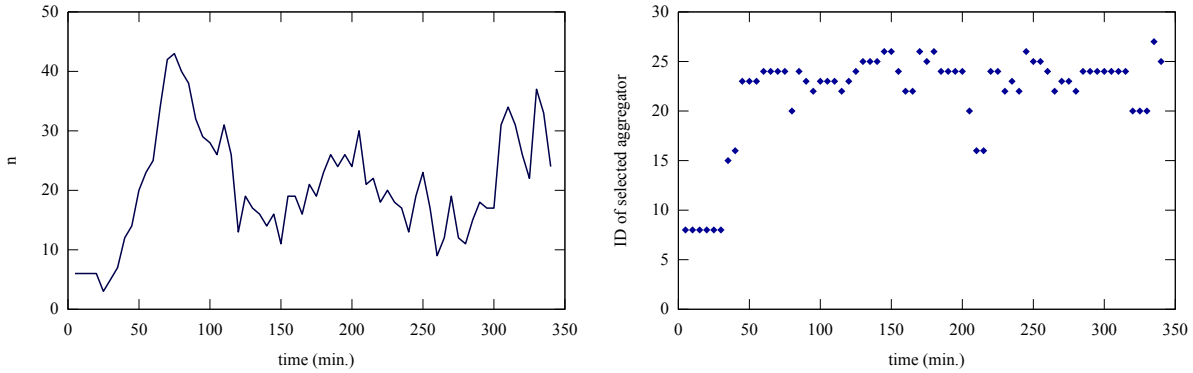


Figure 3.8: False positives of all aggregators (thin lines) and that selected (thick line).



(a) number of challenges (b) selected aggregators

Figure 3.9: Evolution of CAMNEP with adaptation at runtime.

The traffic trace on which the experiments were run had been recorded on a university network and had a length of almost 350 minutes. The flows in the trace were manually labeled as malicious or legitimate. The malicious flows covered mainly scanning and peer-to-peer activities, bot propagation and brute force attacks on passwords. The recording was made in batches of five minutes each. Each batch contains up to ca. 80,000 flows, with roughly ten percent being malicious. In the following, we use the terms *false positives* (FP), *false negatives* (FN) and *true positives* (TP) (see Sect. 2.5.2 for an explanation); these terms imply that a threshold, which decides when to raise an alarm, has been applied to the final output. For more details on how to choose these thresholds we refer to [120].

Adaptive Number of Challenges Figure 3.9(a) shows the number of challenges fluctuating over time. For the first three batches (or 15 minutes), we fixed the number of challenges, because the anomaly detection mechanisms require some time to initialize. After this, the number of challenges increased

Table 3.3: Performance for different fixed aggregators, and the runtime adaptation.

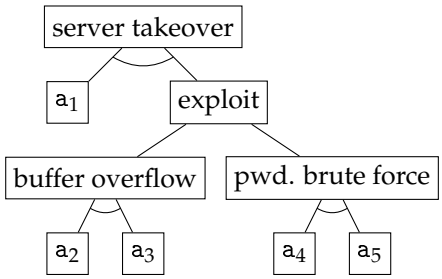
System	FN (Avg., #sources)	FP (Avg., #sources)
Arithmetic average (on anom. det.)	14.7	12.5
Arithmetic average (on aggreg.)	13.1	24.3
Aggregator with min. #FP	14.5	5.3
Aggregator with min. #FN	9.8	125.2
Best aggregator	13.7	5.7
Aggregator selection at runtime	**14.0**	**3.1**

until minute 70 where it peaked, thereafter fluctuating around a central value. One can observe two further increases after minute 150 and minute 300. When comparing these observations to Figure 3.8, which shows the performance of the aggregators, an increase in the diversity of the performances can be observed in these areas. Besides this, the figure shows that a peak in the number of false positives from all aggregators at minute 100 could not be avoided. It can be argued that at this point in time the enhanced anomaly detection mechanisms are not yet sufficiently trained. A big peak later in time, at minute 190, is successfully avoided by the selected aggregators. Figure 3.9(b) shows which aggregator was selected at what time. It is interesting to see that the system mainly selected aggregators from a smaller subset of aggregators, and especially aggregators 23 and 24, which used a combination of weighted- and order-weighted averaging.

An overview of the results is given in Table 3.3. The table shows the number of unique source IP addresses for the classes FN and FP respectively, averaged over the different batches of five minutes each. We can see that the proposed aggregator selection at runtime clearly outperformed the trivial aggregation function, the arithmetic average; but also all other aggregators performed worse either in terms of FP *or* FN. The performances of the aggregator with the minimum number of FP and the best aggregator came close to the performance of the aggregator selection at runtime. However, what these best aggregators were, could only be known after the system was run. Therefore, these represent only best case performances for a fixed aggregator. The better performance can probably be explained by the fact that peaks like those of Figure 3.8 can sometimes be avoided. The results suggest that our approach is able to combine the diversity of the different anomaly detection methods in a beneficial way.

Threat-Model Driven Challenge Selection To test the approach of threat-model driven challenge selection, the attack tree shown in Figure 3.10 was used to determine the composition of challenges. Appendix C shows how the damage for the respective attack classes can be computed in this case.

In a subsequent step, standard security tools such as `nmap` or `metasploit` were used to attack the network in order to realize the corresponding threat. The attacks were repeated several times with changes in speed, tool settings and intensity. The following results suggest that the system selected good aggregation functions, i.e., those that were able to detect various stages of server compromise attacks. Table 3.4 shows the deviation of the attack scores of the malicious flows from the average attack score as a multiple of the standard deviation of all attack scores: Higher values are better, and negative values indicate that the flows were thought to be less malicious than the average. The attack scores



Attack	Description	Attack class
a_1	horizontal scan	A_1
a_2	fingerprinting	A_2
a_3	buffer overflow	A_3
a_4	SSH pwd. bf. request	A_4
a_5	SSH pwd. bf. response	A_4

Figure 3.10: Server compromise attack tree with attack descriptions.

Table 3.4: Attack score deviations for request/response traffic.

Attack	All challenges	Selected challenges
Horizontal scan	1.1/-0.2	1.4/0.0
Vertical scan	1.2/-0.2	1.4/0.3
Fingerprinting	1.5/1.2	1.9/1.6
SSH pwd. brute force	-0.2/0.6	0.2/1.2
Buffer overflow	-0.2/0.1	0.2/0.0

attributed to horizontal scans, fingerprinting and vertical scans have improved considerably, making them far more likely to be detected. The most dramatic change of behavior was related to the brute force password breaking attempts. These were undetectable with the baseline system configuration, but became detectable with the threat-model driven system adaptation. Buffer overflow attacks still could not be detected, however: anomaly detection methods are generally not good at detecting this kind of attack because of the low volume of traffic involved.

Table 3.5: Effects of a threat-model driven challenge selection.

Result	#FN (Avg.)	#FP (Avg.)	#TP (Avg.)
Random challenge picking	39	201	146
Selective challenge picking	37	249	161

The results shown in Table 3.5 suggest that the overall accuracy of the system suffered somewhat from the adaptation. The higher number of false positives is probably caused by the system's focus on the server takeover threat, which was uncommon in the traffic trace. Still, the number of false negatives did not change significantly, and the number of true positives slightly increased.

Chapter 4

Modeling Trust in Information Sources

In this chapter, we show how the trustworthiness of an information source can be assessed in cases where acquired information cannot be verified directly. These assessments can be used as criteria for making decisions about which information provider to select next. In addition to this, acting as an extension to the spot-checking mechanism from the previous chapter, it helps to reduce the overhead of spot-checking: The more one trusts, the less one has to check. Since the trust model has several parameters, we also show how these may systematically be configured.

Organization In Section 4.1, we investigate general challenges and limitations for computationally modeling trust. We then study specifics of trust in the context of information acquisition, and provide a definition. In Section 4.2, we propose and evaluate a concrete evidence-based trust model that assesses the trustworthiness of information sources. Finally, we show in Section 4.3 how to find optimal configurations of evidence-based trust models, and provide a proof of concept by applying it to the proposed trust model.

Terminology Following the common terminology, we call an entity that trusts "trustor", and an entity that is trusted "trustee".

This chapter is related to [150, 151, 149] and [145, 144, 148].

4.1 Computational Trust Modeling

In the following, we derive a definition of *trust*, and discuss the problems we face when formalizing it as a computational concept. We then examine the specifics of trust in information sources.

4.1.1 Trust

Trust plays an important role in the social behavior of humans. In short, it helps a human to make decisions about relying on other humans. When a trustor relies on a trustee, he has certain expectations about the trustee's behavior. In market scenarios, these expectations can for instance be defined in form of a contract. Then, the trustee either meets the expectations, which we describe as "the trustee succeeds", or he disappoints the trustor, described as "the trustee fails". When a trustor has decided

to rely on a certain trustee, and has asked him to perform some task, the trustee will do either of the following:[1]

- accept, and succeed
- accept, but fail
- reject

We can see that trust concerns the belief that someone will succeed under the condition that he will accept. If someone rejects a request, and also if he does it very often, this does not mean that he is untrustworthy; he might have foreseen in every case that he would not have been able to succeed – and this is actually honest behavior. Note that a trustor might nevertheless stop asking someone who rejects requests too often – but such a decision is basically not about trust.

To succeed, a trustee has to be capable, so trust is clearly, among other things, about the skills of an entity. However, consider the division of a trustee's abilities into:

- his skills,
- his ability to assess his own skills.

A trustee that underestimates his skills, will probably not accept a request; but when he agrees, he will probably succeed, and so he will appear trustworthy. A trustee that overestimates his skills will probably agree to a request and fail. This means that trust is actually more about the ability to assess one's own skills than about the skills themselves.[2] The self-assessment of skills implies another important ability that a trustee should have: the ability to predict the environment. Only if a trustee is able to predict the environmental conditions for his actions, he can assess whether he will be able to succeed.

Finally, since a trustee is autonomous, even if he is skilled and able to assess his own skills, the trustee can also accept and intentionally fail. Reasons for an intentional failure can be to save costs or to harm the trustor. We call this kind of behavior *malicious behavior*. For humans, the unwillingness to perform, due to a lack of motivation, can also be the reason for a failure; however, we will not take this cause into account, because it is currently not relevant for machines.

To summarize, "to trust" means generally to believe both in the skills of the trustee, which encompasses his ability to correctly predict whether he will be likely to succeed, and in his willingness. Many problems arise in the process of building trust, even in the very early stages of collecting evidence. In the following, we discuss such problems.

4.1.2 Modeling Trust

Much research has tried to adapt trust, as we humans use it, to the world of computers. To this end, a formal computable model has to be proposed. Several approaches in this direction have been pursued (e.g., see [117]). We consider the approach of *evidence-based trust modeling* as discussed in Section 2.3.3.1.

Evidence-based trust models specify how to derive trust in an entity from a set of past experiences with that entity. Positive experiences generally increase trust, and negative experiences reduce it. To achieve this, a model commonly follows the following three steps (see also Fig. 4.1):

[1] Following the restriction to correct/incorrect information (see Sect. 1.1.2), we also restrict ourselves to the binary view of failure and success of a trustee.

[2] There is evidence that at least a human's ability to assess his own skills strongly depends on his skills [89]; unskilled people tend to overestimate, whereas skilled people tend to underestimate their skills.

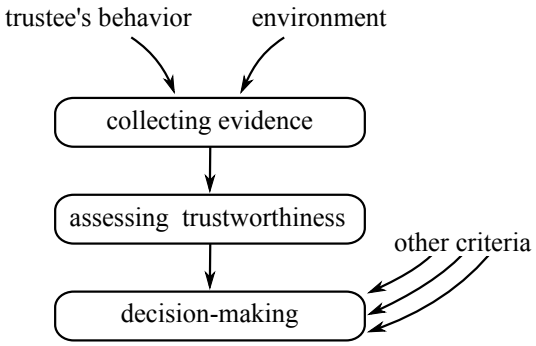


Figure 4.1: Stages of evidence-based trust reasoning.

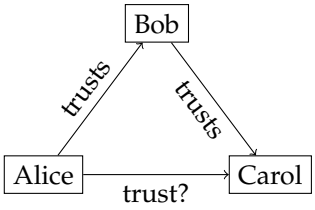


Figure 4.2: Transitivity of trust.

1. Collect evidence from interactions with other entities, in particular distinguishing positive from negative experiences.

2. Combine the collected evidence to get an accurate assessment of an entity's trustworthiness.

3. Use the resulting trust as one criterion in decision-making concerning possible interaction with other entities.

4.1.2.1 Properties of Trust

In general, trust is *not transitive*; this is illustrated by the following example (see also Fig. 4.2): The fact that Alice trusts in Bob and Bob trusts in his mother Carol does not imply that Alice should trust in Bob's mother as well. Still, it is justified to apply transitivity of trust in this case, if the following conditions are fulfilled:

1. Alice trusts in Bob to not lie.

2. Alice trusts in Bob's assessment of Carol's trustworthiness.

3. The *relations* (Carol, Bob) and (Carol, Alice) are "equal".

The fuzzy term "equal" refers to the fact that Carol might be variably trustworthy to different persons. In the above example, Carol might for instance behave trustworthy towards Bob, because he is her son, but not trustworthy towards Alice, because she is not a member of her family. Consider another example, which is also covered by Figure 4.2. Carol is a politician, Bob is her husband, and Alice is a reporter. Even if Bob can truthfully tell Alice about his justified trust in Carol (conditions 1 and 2 are fulfilled), this does not imply that the reporter can trust in what Carol says; this is the case because the third condition is not fulfilled. In the scenarios that we are addressing in our thesis, that is where computers interact with each other, the third condition concerns for instance the relations between the people operating the computers.

From the above examples, it directly follows that trust is *subjective*, because it might be the case that Bob trusts Carol, while Alice does not. Furthermore, trust is *not symmetric*, that is, the fact that Bob trusts Alice does not imply that Alice trusts Bob. Finally, trust is *context-sensitive*; to give an example, one can trust someone to drive a car and at the same time not trust that person to fly a plane. This issue is discussed in more detail in Section 4.1.2.4 below.

4.1.2.2 Sources for Trust

There are several ways to collect evidence for assessing an entity's trustworthiness. One way, and this is the only one we consider in our work, is to extract evidence from *direct experiences* with that entity. If for instance Bob has sold low-quality products to Alice several times, Alice can see this as an indicator of the general quality of products sold by Bob, and so will no longer trust in Bob in this regard.

To build trust, one can also rely on information concerning trust that comes from others. In the basic case, other entities report on the *direct experiences* they have had with the entity in question. They can also report about the *trust* they have in that entity, for instance in the form of a *recommendation* for an entity. Whenever one uses information coming from others in the process of building trust, one has to be careful, because as discussed above, trust is in general subjective and not transitive. Besides this, entities may also report incorrect or unreliable information [180]. However, especially when an entity has just entered a system and has not yet enough direct experiences with other entities, it may be helpful to rely on the opinions of others. In order to mitigate the effects of misreporting entities and the subjectivity inherent in trust, a "reputation system" [125] can be used that combines the opinions of a large group of entities about a single entity's trustworthiness. Such a combined, global, assessment of trust, is called the entity's "reputation" among that group. To provide meaningful results, these systems require that many entities submit their trust assessments of other entities to the reputation system.

In our work, we only consider direct experiences, because there are scenarios where this is the only source for trust, e.g., in volunteer computing systems. As a side effect, this implies that we do not have to deal with the problematic transitivity property of trust.

4.1.2.3 Delusive Evidence

"Both types of mistake – trusting too well and not well enough – can be costly." [51]

This quotation leads to the question of what observation is factual evidence for a trustee being more or less trustworthy. In particular, whether a success is always evidence for a higher trustworthiness, and vice versa, whether a failure is always evidence for a lower trustworthiness. Consider the following example of an aviator:

Example 4.1. *Aviator Howard is considered as one of the best aviators, and you trust him too. So, you rely on him and let him fly you to some destination. There is no evidence of bad weather, and so he predicts that the weather will be fine. He takes off and the flight appears to become routine. But suddenly, a storm whips up from nowhere. As a consequence, the plane crashes – but he and all the passengers miraculously survive. He did not succeed in flying the plane to the destination. However, his trustworthiness as a pilot should not decrease; it could even increase.*

Prompted by this example, we now analyze in which cases the trust in another entity should be decreased and when it should be increased. This will show that there are both cases where a failure does not form negative evidence, and where a success does not form positive evidence.

The following reasoning is illustrated in Figure 4.3. After having selected a trustee, a trustor requests the trustee to perform a certain task. The trustee has then to decide whether to accept the request or not. If he is malicious, he will subject this decision to his strategy; in the case that he accepts, his strategy will further determine whether he is going to fail intentionally. Generally, malicious intent in a trustee is sufficient reason to decrease the trust in the trustee. Of course, this malicious intent cannot always be recognized, particularly if the trustee pretends to be honest, but this analysis tries only to

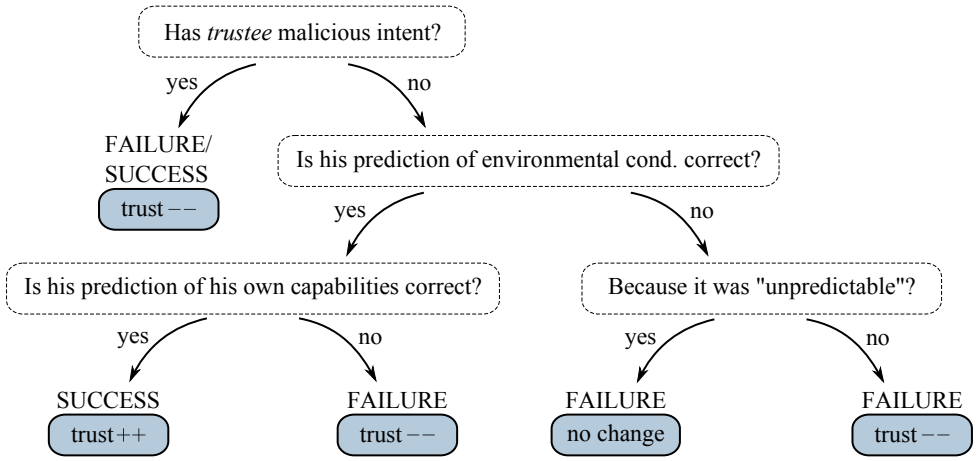


Figure 4.3: When trust should be increased ($++$) and decreased ($--$).

identify the different cases without discussing how to distinguish them in practice. If the trustee is not malicious, he will try to predict whether he can successfully handle the request. To this end, he forecasts the environmental conditions, and assesses his skills for processing the request under these expected circumstances. He only accepts if he believes that he will succeed. However, the actual environmental conditions can differ from the trustee's prediction and he might fail for this reason. The error can be due to the trustee's inability to make this prediction; but the conditions can also evolve in an "unpredictable" way, in which case his failure should be considered *excusable* [148]. The meaning of "unpredictable" needs to be decided by the trustor; it depends among other things on how much time passes between the acceptance and the processing of the request, and on whether other trustees would have been capable of making the correct prediction. Another reason for a failure of the trustee is the trustee's overestimation of his capabilities. This can even be the case if the prediction of the environmental conditions is correct. Since this misestimation will probably occur for future requests as well, the trust should be reduced. Again, see Figure 4.3 for an overview of this reasoning.

To make the boundary between positive and negative evidence as clear as possible, it is important to explicitly state in a contract the environmental conditions under which a trustee is expected to succeed. However, malicious trustees can still delude a trustor with a strategic success, and a failure can still be excusable if the environment evolves in a manner not foreseen by the contract, what is also known as "unknown unknown".

A Limit of Trust Modeling The delusiveness of evidence implies one important limitation of trust modeling. Given that the trustor is not able to look in a trustee's mind; *if a malicious trustee is able to emulate the behavior of a trustworthy trustee, we cannot distinguish between a trustworthy trustee and a malicious trustee, who pretends to be trustworthy.* Basically, we face the halting problem [140] or the problem of verifying a scientific theory. This suggests that for a trustee that has never showed malicious behavior up to the present, we cannot make any conclusion. Nevertheless, there is one reason why we do learn something about a trustee who always behaves in a trustworthy manner. In scenarios where trustees have incentives to behave maliciously, trustworthy behavior entails costs (such as the time expended in building trust). Therefore, a malicious trustee that behaves trustworthily is making an investment, and wants to break even at some point, which in turn requires malicious behavior in the future. This argues in favor of treating trustworthily behavior as evidence for trustworthiness. Actually, if the costs of a

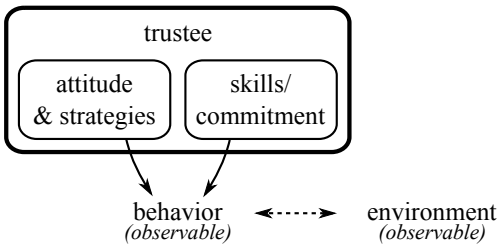


Figure 4.4: The mind of the trustee is not observable.

trustee for behaving trustworthy and maliciously could explicitly be modeled, trust could be defined as: the difference between the investment of the trustee in the past, and the profit he could make when behaving maliciously in the next interaction multiplied by the probability of being exposed.

4.1.2.4 Further Issues

In the previous paragraph, we have seen that it is not obvious whether an outcome of an interaction is actually positive or negative evidence. The intrinsic reason for this is that the trustor has no access to the trustee's mind, as illustrated in Figure 4.4. At the most, the trustor can observe the behavior of a trustee and (parts of) the environment in which the trustee acts. However, in order to predict a trustee's behavior in future interactions, the trustor needs to understand the trustee: his attitude, his strategic intents, his abilities and his commitment. A sophisticated trust model would account for all these characteristics of the trustee. This emphasizes that trust modeling deals with issues that differ from those addressed by conventional forecasting techniques such as extrapolation or trend estimation. In the following, we discuss several such issues.

Whitewashing In many distributed systems such as P2P systems, the identities of entities are often not linked to physical resources. As a result, entities can take part in the systems with several distinct identities [40]. In these scenarios, *whitewashing* [112, 71] becomes possible: An entity can try to change its identity in order to reset its damaged reputation. To avoid this happening, a trust model has to make the initial trustworthiness assigned to an identity low enough that an entity cannot profit from renewing the identity. The threat of whitewashing is addressed in Section 4.2.2.1, where we evaluate our trust model.

Context-Sensitivity Trust can be called *context-sensitive* if an experience that has been made in a certain context is only significant for trust-based decisions in similar contexts. McKnight and Chervany [101] give the example that "one would trust one's doctor to diagnose and treat one's illness, but would generally not trust the doctor to fly one on a commercial airplane". So, a *first* notion of context means the *type of task* that has to be performed [68, 85]. In a *second* notion, context refers to the *circumstances* under which a task has to be performed [119]; for instance if a pilot succeeds in flying a plane in a storm, he will be able to fly the plane in good weather, too – but the converse is not that obvious. Generally speaking, *everything* that can impact the behavior of a trustee and is not part of the trustee itself, belongs to the context [38].

The context-sensitivity of trust can be exploited by attackers, for instance to launch an attack on a volunteer computing system as follows.[3] For a series of very easy computational tasks the attacker re-

[3]The authors of [84] call such attacks "value imbalance attacks".

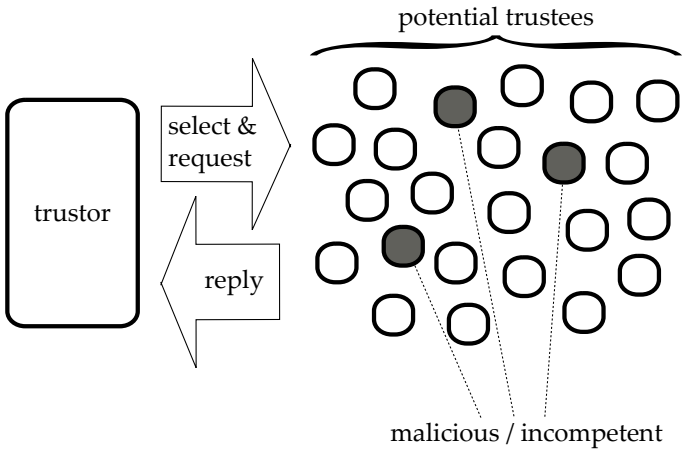


Figure 4.5: Situation where trust is helpful.

turns the correct results, but then for a very complex task, he returns a random result. This attack shows that a number of correct results for easy tasks is not necessarily evidence for this worker's performance on complex tasks.

The spot-checking mechanism proposed in the previous chapter is also part of the trust model that is proposed later in this chapter. For this mechanism, to make sure that the probabilities of single requests within a set of requests are independent, all requests have to belong to the same context. To account for context-sensitivity beyond a single set of requests, other mechanisms have to be used. Instead of explicitly addressing this issue in our work, we refer to existing approaches from the literature: [119, 32, 86].

Exploration vs. Exploitation Figure 4.5 shows how a distributed information system appears from the perspective of a trustor (analogous to Fig. 1.1). Consider the situation where the trustor has a sequence of tasks that he wants do delegate one after another. Furthermore, assume that for each delegated task, the trustor can collect evidence about the respective trustee, e.g., whether he succeeded or failed. At the beginning, the trustor chooses some trustee *randomly*, because he has no knowledge about the entities. After this, the trustor learns about the trustworthiness of the selected trustee. At this point, the question is how to select the next trustee. Assume the selected trustee succeeded, meaning that he might be seen as more trustworthy than all other entities. Still, he might in fact be less trustworthy. This results in a dilemma known as the "exploration vs. exploitation"-problem: The trustee has to *explore* the population of entities to get to know them better; but at the same time, the trustee wants to *exploit* his knowledge about the entities' trustworthiness and select only those that appear trustworthy. One basic approach to this problem is known as the "Boltzmann exploration", in which one selects an entity with a probability that is proportional to the value that this selection is expected to have [113, Chapter 10]. We adapt this approach later in Section 4.2.4 to the use in our trust model.

The explore vs. exploit dilemma is known in the areas of approximate dynamic programming and reinforcement learning. In addition to these domains, the dilemma gains an additional facet when it concerns the entities' trustworthiness, namely the aspect of *forgiveness*. Suppose that a trustor selects a trustee, but the trustee fails because of a lack of commitment. By purposefully selecting this trustee again, the trustor shows the trustee that he is prepared to give him another chance, which in turn can motivate the trustee to make a greater effort. However, this aspect, which is strongly related to psychol-

ogy, is less relevant for our work, because we consider only malicious and honest machines.

4.1.3 Trust in Information Sources

We define *trust* in the context of information acquisition as follows:

Definition 4.1 (Trust in an information source)**.** Trust *is a belief of an entity A about an autonomous source B, which helps A to estimate whether a reply $\mathcal{R}$ provided by B upon some request $\mathcal{Q}$ by A, makes $(\mathcal{Q}, \mathcal{R})$ a fact, without A having to check $\mathcal{R}$. Trust anticipates both B's attitude towards A, and B's ability to ensure that the provided information is correct.*

This implies that an entity A trusts maximally in an information source B, if it believes that:

- if B provides information $\mathcal{R}$ to some request $\mathcal{Q}$ from A, then B strongly believes that $(\mathcal{Q}, \mathcal{R})$ is a fact, and
- if B strongly believes something, then it is correct.

The first belief concerns B's *intentions*: If B were malicious, it would probably provide some $\mathcal{R}'$ believing that $(\mathcal{Q}, \mathcal{R}')$ is *not* a fact, or it was not sure whether $(\mathcal{Q}, \mathcal{R})$ is a fact. The second belief concerns B's *competence*, i.e., if B were *incompetent*, its belief in $(\mathcal{Q}, \mathcal{R})$ would not be founded or wrong. Note that this definition conforms with the notions defined by Demolombe [36] (see also Sect. 6.1.3.1 for a discussion).

4.1.3.1 Incompetent and Malicious Sources

As mentioned above, there can be incompetent or malicious information sources. An *incompetent* but honest information source is not able to guarantee that the information it provides is correct, but thinks that it is able to do so; apart from this, it is not lacking of commitment nor has it malicious intent; in fact, it aims to provide the truth. In contrast to an incompetent source, a *malicious* source does not aim to provide the truth. We distinguish four different types of malicious information providers:

Careless Source Does not care about the truth.

Cheater Tries directly to maximize his profit.

Manipulator Provides information in order to guide the trustor in a certain direction.

Saboteur Tries to provide incorrect information so as to harm the trustor.

Malicious sources have in common that they try to stay undetected by the trustor – they want to stay in the game as long as possible. A *careless* information source provides information although it neither believes that the provided information is correct nor that it is incorrect. Reasons for this behavior can be for instance that the source either is able to ensure the correctness of provided information, but is not committed enough to do so, *or* that the source knows that it is not able to ensure the correctness, but provides the information anyhow. Frankfurt [49] calls these information providers "bullshitters" (see also [27]). The three other types of malicious sources are generally both competent and committed, and so intentionally provide incorrect information. A *saboteur* tries to maximize the damage he can inflict on the trustor. In contrast to the saboteur, *cheaters* and *manipulators* do not necessarily want to harm the trustor: Cheaters try directly to optimize their profit, for instance by providing cheaply generated information that is known to be incorrect; manipulators provide incorrect or incomplete information

with the aim of causing the trustor to make decisions based on this information, which are in the end beneficial to the manipulators – but possibly harmful to the trustor.

For the evaluation of our work we focus on incompetent sources and cheaters, because careless sources and manipulators seem to be irrelevant in computer networks; the evaluation of our work for saboteurs is discussed in Section 6.2.3. The difference between incompetent and malicious sources is illustrated by the following two examples.

4.2 Evidence-Based Trust in Information Sources

Trust can be helpful in ensuring the correctness of acquired information in cases where verification of the information is not easily possible; as outlined in the introduction, the verification can for instance be too costly or time-consuming; or it can only be carried out at a later point in time; or cannot be done at all. However, in precisely these cases, it is not clear how to assess the trustworthiness of an information source: The provided information cannot be verified, and so cannot be used as evidence of a source's trustworthiness. At this point, spot-checking with challenges from Chapter 3 can be used: Replies to challenges *can* be verified, and so can serve as evidence. In turn, the more certain we are that a source is trustworthy, the fewer challenges we have to use for spot-checking. Note that the pure spot-checking mechanism is concerned with estimating the correctness of acquired *information*, whereas the trust model proposed in the following concerns the properties of an *information source*.

4.2.1 Basic Model

In the following, we describe a model to compute a "trust value", that is, a measure of an information provider's trustworthiness. An additional value of *uncertainty* measures how significant the trust value is. These two values can later be used as criteria for selecting good information sources.

The model assumes that an information source can be described by a probability ρ: With probability ρ it returns incorrect information to a single request, with probability $1 - \rho$ it returns the correct information. We call ρ also the *error rate* of an information source. The model is based on a repetitive application of the spot-checking mechanism from the previous chapter. The aim of the model is to extrapolate the estimates of past error rates of an information source, while keeping in mind the possibility of malicious sources that intentionally increase or decrease their error rate over time. One main idea of the model is to group sequential evidence together as long as it is self-consistent. If new evidence comes up that conflicts too much with older evidence, it is assumed that the error rate of the provider has changed, and so the older evidence is treated separately. This accounts both for tactical behavior and also for changes in a provider's competence.

4.2.1.1 Single Error Rate Estimation

Applying the spot-checking mechanism from the previous chapter, the number of incorrect replies to a set of n challenges is binomially distributed with parameters n and ρ. So, we can estimate the error rate ρ of the information source with the use of Bayes' theorem:

$$f(\rho|y) = \frac{p(y|\rho)f(\rho)}{p(y)} \,. \tag{4.1}$$

The probability of finding exactly y incorrect answers to the challenges, given ρ, follows from the basic product rule [77, p. 51] and the independence between the single replies:

$$p(y|\rho) = \rho^y(1-\rho)^{n-y} \,. \tag{4.2}$$

Since we are dealing with attackers, we again choose the prior, which is continuous here, to be uniform:[4]

$$f(\rho) = \begin{cases} 1 & \text{for } 0 \leq \rho \leq 1 \\ 0 & \text{otherwise} \,. \end{cases} \tag{4.3}$$

The probability $p(y)$ can be computed by marginalization over ρ:

$$p(y) = \int p(y|\rho) f(\rho) d\rho \tag{4.4}$$

$$= \int_0^1 \rho^y(1-\rho)^{n-y} d\rho \,. \tag{4.5}$$

Putting everything together, what we eventually find is the *beta distribution* [170, 171]:

$$f(\rho|y) = \frac{\rho^y(1-\rho)^{n-y}}{\int_0^1 \rho^y(1-\rho)^{n-y} d\rho} \,. \tag{4.6}$$

The beta distribution is defined by two "shape parameters" α and β, which in our case are $y+1$ and $n-y+1$ respectively. The mean of the beta distribution, and therefore the best estimator $\hat{\rho}$ of the error rate, is:

$$\hat{\rho} = \frac{\alpha}{\alpha+\beta} = \frac{y+1}{n+2} \,. \tag{4.7}$$

The standard deviation of the beta distribution is:

$$\sigma = \sqrt{\frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}} \,. \tag{4.8}$$

4.2.1.2 Trust Values

We base our assessment of an information provider's trustworthiness on the information we have to hand, namely how accurate the source was estimated to be in the past. Table 4.1 shows an example of such information. Each row represents an application of the spot-checking mechanism from the previous chapter. One request/reply is described by a number of real requests (m) and challenges (n), and the number of incorrect replies to the challenges (y). Based on n_i and y_i, the error rate $\hat{\rho}_i$ of the information source can be estimated for each request. In order to learn from the past, we want to combine these error rate estimates into an aggregate *trust value*. When no request has been made yet, we create an initial error rate estimate $\hat{\rho}_0$ for $n = y = 0$, which gives $\hat{\rho}_0 = 0.5$. This estimate reflects our a priori trustworthiness assessment of an information source. For a new reply from a source, we then compute

[4] For a discussion of this issue see Section 3.3.1.1.

Table 4.1: History of requests made to one information provider.

Request ID	#Real requests	#Challenges	#Incor. replies to the challenges	Error rate estimate
$i = 1$	$m_1 = 10$	$n_1 = 10$	$y_1 = 3$	$\hat{\rho}_1 = \frac{y_1+1}{n_1+2}$
$i = 2$	$m_2 = 5$	$n_2 = 20$	$y_2 = 1$	$\hat{\rho}_2 = \frac{y_2+1}{n_2+2}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$i = 99$	$m_{99} = 10$	$n_{99} = 10$	$y_{99} = 2$	$\hat{\rho}_{99} = \frac{y_{99}+1}{n_{99}+2}$

an aggregate *trust value* t for that source in form of a weighted average:

$$t \stackrel{\text{def}}{=} \frac{\sum_i w_i(1-\hat{\rho}_i)}{\sum_i w_i} , \tag{4.9}$$

where the w_i are positive weights that can be used to give older estimates $\hat{\rho}_i$ less weight than newer. The trust value t, and also the variants that are proposed below, are real values in the interval $(0, 1)$. The values 0 and 1 are not possible, because all error rate estimates $\hat{\rho}_i$ are in $(0, 1)$ as well. This corresponds with the intuition that an arbitrarily long sequence of positive experiences with a source is no guarantee that a negative experience will never occur, and vice versa.

Choice of Weights The only restriction we impose on the weights is that at least one of them has to be greater than zero. How fast the weights decrease defines how fast the trustor thinks that older evidence becomes outdated; in other words, how fast the trustor "forgets" or "forgives". To not forget at all, all weights can be set to 1. To get linearly decreasing weights, they can be defined as $w_i \stackrel{\text{def}}{=} (i + 1)$. To make a linear moving average that discards all but the l last replies, they can be defined as $w_i \stackrel{\text{def}}{=} \max(i - i_{\max} + l, 0)$, where $i_{\max}$ is the index of the latest request. Finally, to apply an exponential moving average, the above trust value can be computed recursively with the use of an "aging factor" $A \in (0, 1]$:

$$t_i \stackrel{\text{def}}{=} A \cdot (1 - \hat{\rho}_i) + (1 - A) \cdot t_{i-1} . \tag{4.10}$$

This update rule is commonly used in the field of reinforcement learning [155].

Combining Evidence of Multiple Replies Assume that ρ does not change over time. Then, the larger the number of challenges in an estimation of the error rate $\hat{\rho}$, the more precise the estimate will be. Therefore, combining the evidence first and then computing an error rate estimate yields a different result than computing the error rate estimate for small portions of evidence first and then combining them as in eq. (4.9); this is because in the latter case several imprecise estimates are combined, which makes the resulting estimate no more precise. However, the combination of evidence is justified probabilistically only if ρ does not change over time; but provided that the error rate estimate changes only little from one reply to another, the probability is high that the actual error rate has changed only little as well. Therefore, it makes sense to group evidence together as long as it is similar. If an abrupt change in the error rate occurs, a new group of evidence is created. What is more, the grouping of similar evidence attaches special attention to abrupt changes in the error rate estimates. At the same time, this makes the

model more robust against attackers, as we show later in the experiments, because abrupt changes are especially characteristic of malicious information providers that intentionally change their error rate, for instance due to strategic considerations.

The idea of combining evidence can be implemented as follows. Let $\hat{\rho}[l_1, l_2]$ denote an error rate estimate based on the combined evidence from the replies with indices l_1 to l_2, i.e., based on $\{\hat{\rho}_i\}_{l_1}^{l_2}$. A threshold ϵ, which we call the "tolerance factor", is now defined when the evidence of two or more consecutive replies is combined: If there is an x such that all consecutive error rate estimates $\{\hat{\rho}_i\}_{l_1}^{l_2}$ are in the interval $[x-\epsilon, x+\epsilon]$, then their evidence is combined to compute $\hat{\rho}[l_1, l_2]$. We call $[x-\epsilon, x+\epsilon]$ the "tolerance interval"; the tolerance interval actually determines how much variation in the correctness of the information should be tolerated; one has to keep in mind though that such variations are partly caused by the fact that the challenges are randomly placed in the request.

The algorithm for combining the evidence works as follows. For every new error rate estimate with index l_{new} check whether it is similar enough to the latest group of error rate estimates, and if so, combine the evidence; otherwise:

1. Search for the earliest l such that $\{\hat{\rho}_i\}_l^{l_{\text{new}}}$ respects some tolerance interval $[x-\epsilon, x+\epsilon]$; possibly the earliest such l is l_{new} itself, and $\{\hat{\rho}_i\}_l^{l_{\text{new}}}$ consists only of $\hat{\rho}_{l_{\text{new}}}$.

2. Repeat step 1 starting from index $l-1$ downwards, creating further groups, until all evidence has been grouped.

For each set of combined evidence with indices from l_1 to l_2, compute a combined error rate estimate based on y_{l_1} to y_{l_2} and n_{l_1} to n_{l_2}. Lastly, compute the final trust value, denoted by $\dot{t}$, by applying eq. (4.9) to the set of combined error rate estimates. In contrast to t, this trust value $\dot{t}$ accounts for sudden changes in an information provider's error rate, which point to strategic moves, and thus to malicious behavior.

4.2.1.3 Uncertainty

A trust value does not reflect how certain we are about its significance. For instance, when no information has been requested from a particular information source, the trust value for the source is set to $t_i = 0.5$ (because we have $t = \frac{w_0 \cdot (1-\hat{\rho}_0)}{w_0} = 1 - \hat{\rho}_0$). This value is caused by the assumption of a uniform error rate prior, but it is not backed by any evidence. If now an observation of y incorrect and equally many correct challenges is made, the trust value will stay at 0.5, but this value is now backed by some evidence. Therefore, we additionally compute a *measure of uncertainty* that reflects the amount of evidence used for the computation of t. Wang and Singh [167] postulate that a measure of uncertainty should have two properties:

- the more evidence is taken into account, the lower is the uncertainty, and
- the more "conflict"[5] there is in the evidence, the higher is the uncertainty.

We prove in Appendix A.3 that the standard deviation of the beta distribution has both properties. Since the standard deviation is simpler and easier to compute than the measure introduced by Wang and Singh [167], we use it as our uncertainty measure.

Let σ_i be the standard deviation corresponding to $\hat{\rho}_i$. To normalize it to the interval $(0, 1]$, we use $2\sqrt{3}\sigma_i$ (see App. A.3). To get the uncertainty corresponding to t we apply the weighted average as for

[5] Conflict is high if the number of correct and incorrect replies is similar.

the trust value:

$$u \stackrel{\text{def}}{=} 2\sqrt{3} \cdot \frac{\sum_i w_i \sigma_i}{\sum_i w_i} \ . \tag{4.11}$$

If no evidence is given, the uncertainty takes value 1. If we get the first reply, we can make our first error rate estimate and the uncertainty drops. This is not yet our final uncertainty measure: As long as we do not change the number of challenges, for any further reply the level of uncertainty will *stay at the same level*, because for all new evidence that comes in, we discard the same amount of older evidence. At this point, we use the idea of combining the evidence of several consecutive error estimates. Let n_i be the number of challenges used for reply i, and y_i be the number of incorrect answers to these challenges. Then we can combine the evidence of a set of consecutive replies l_1 to l_2 as follows:

$$\alpha[l_1, l_2] = \left(\sum_{i=l_1}^{l_2} y_i \right) + 1 \ , \tag{4.12}$$

$$\beta[l_1, l_2] = \left(\sum_{i=l_1}^{l_2} n_i - y_i \right) + 1 \ . \tag{4.13}$$

The standard deviation can now be computed on the basis of these shape parameters to give $\sigma[l_1, l_2]$. Above, we showed how to apply the tolerance factor ϵ to shrink the set $\{\hat{\rho}_i\}_i$. We now compute the standard deviations on this reduced set, and combine these as in eq. (4.11) to get the final uncertainty measure, denoted by $\dot{u}$. As a consequence, the uncertainty for a source with constant error rate will drop, because the set of evidence for a particular error rate increases.

The uncertainty measures u and $\dot{u}$ are in the interval $(0, 1]$; in the beginning they both are at 1, decreasing as more evidence comes in, but never reaching 0. This corresponds with the intuition that one can be totally uncertain about something, but never totally certain.

4.2.2 Evaluation of Basic Model

Firstly, the computation of trust and uncertainty values are experimentally evaluated against defined attacker models, and compared to other models from the literature. Secondly, the accuracy of the trust value as an error rate estimate for the current reply is assessed. Finally, the computational complexity of the trust and uncertainty values is analyzed.

4.2.2.1 Evaluation against Different Source Types

Our model is evaluated against the following criteria:

- handling of trustworthy information sources
- dealing with error rates $\rho > 0$
- recognizing sources that behave honestly for a long time and then return some errors ("repetitive cheaters")
- robustness against whitewashing attacks.

We examine the trust value and the attached measure of uncertainty separately. Three particular types of information source are considered:

Trustworthy source An information source that returns an incorrect reply with probability 0.00221. This error rate agrees with the findings of Kondo et al. [88] in the volunteer computing scenario.

Source with $\rho = 0.1$ A source that returns an incorrect reply with probability 0.1.

Repetitive cheater A source that behaves as if it is a honest source, but submits 10 incorrect replies every 10th set of requests.

Trust Value For the computation of trust values, the following four trust models from the literature are considered:

Jøsang & Ismail [79] If normalized to $[0, 1]$, their core "reputation rating" is defined as the mean of the beta distribution $\frac{r_i+1}{r_i+s_i+2}$, where the terms r_i and s_i denote the evidence after interaction i, and are defined as $y_i + A \cdot r_{i-1}$ and $(n - y_i) + A \cdot s_{i-1}$ respectively, with $r_0 = s_0 = 0$; in our simulations, we set the aging factor A to 0.5 and discuss the impact of other values. Since we do not consider evidence coming from other trustors, the rest of their model is not relevant to our comparison.

Zhao et al. 1 [185] Starts with a trust value of 0. For every correct reply the trust value is increased by some constant, and reset to 0 for an incorrect reply. For the simulations, we set the constant to $1/32$ and discuss the impact of other values.

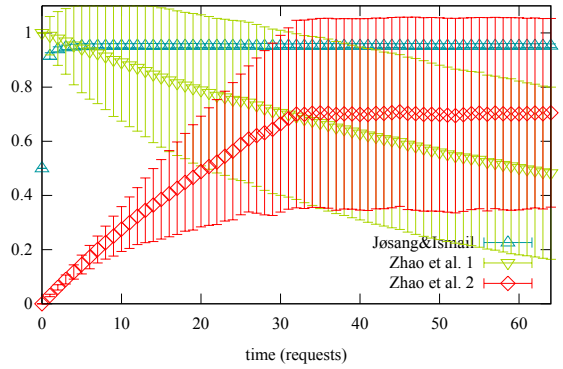
Zhao et al. 2 [185] Starts with a trust value of 1. For every incorrect reply the trust value is divided by two.

Wang & Singh [167] In their model, a core trust value is computed as $\frac{y+1}{n+2}$, which is once again the mean of the beta distribution. To make this into what they call a "trust belief", they multiply the core trust value with some uncertainty value u. For the trust value comparison, we consider the core trust value, and separately compare their uncertainty measure to ours in the next section.

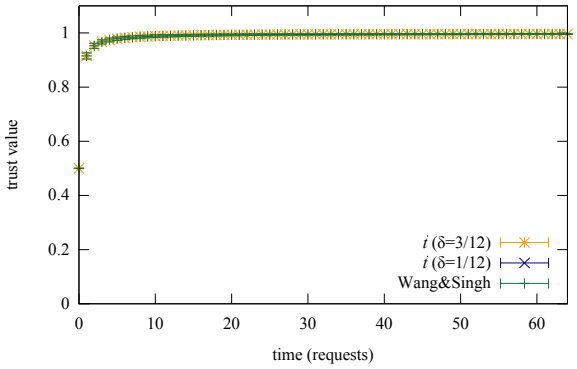
We compare these models to our trust model $\dot{t}$, using $m = n = 10$, with the threshold for combining evidence either set to $\epsilon = 3/12$ or $\epsilon = 1/12$; the first value tolerates a variation of y within some interval $[i - 3, i + 3]$, while the second tolerates a variation within $[i - 1, i + 1]$. The weights are set to $(0.6, 0.3, 0.1)$.

In order to give statistically significant results, the simulations were run 2^{10} times, and average values and standard deviation have been computed. Results are shown in Figure 4.6.

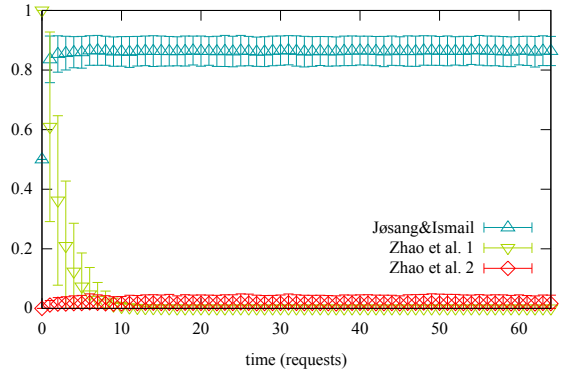
Results The two models of Zhao et al. are not able to cope with error rates of honest sources greater than zero as seen in Figures 4.6(a) and 4.6(c). For Zhao et al. 1, the trustworthiness of a honest source is reduced over time, and the trustworthiness of a source with $\rho = 0.1$ goes to zero very quickly. Similarly, Zhao et al. 2 keeps the trustworthiness of a honest source at some maximum level around 0.7, and a source with error rate $\rho = 0.1$ is considered completely untrustworthy. The reason for this is that upon a single incorrect reply, which happens with a certain low probability for these sources, the trustworthiness is decreased by half, or reset to zero respectively. In addition to that, Zhao et al. 1 lacks a feature for forgetting; that is, once a source has been assessed untrustworthy, this assessment cannot be adjusted, and thus not be corrected if it was inaccurate. In the case of a repetitive cheater this might seem to be the right approach; however, if the source is honest but makes many errors on a single occasion (e.g., for request 10), the model would forever ignore this source, even if it were to be the most trustworthy source thereafter. Zhao et al. 2 handles the repetitive cheater well, because it makes it hard,



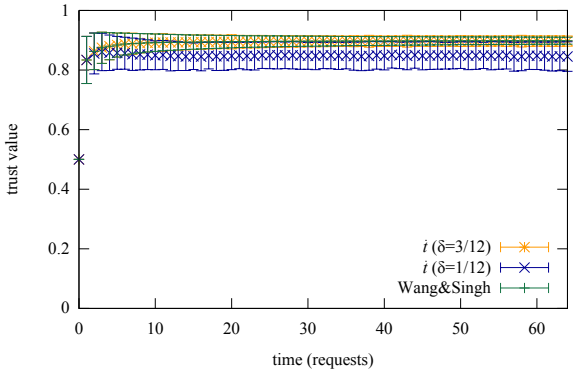
(a) Trustworthy source.



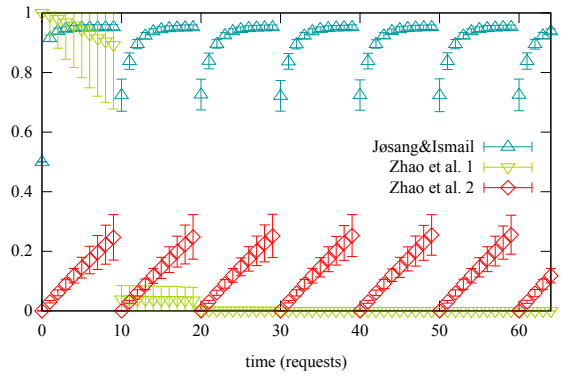
(b) As on the left.



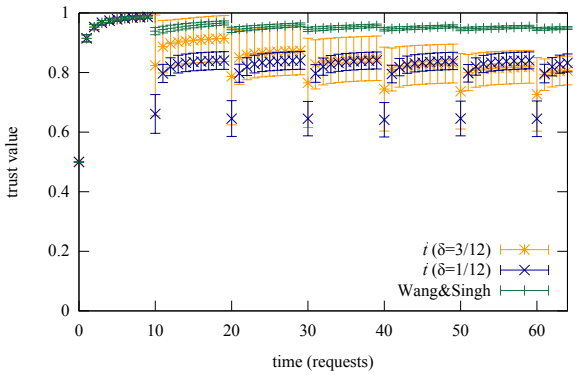
(c) Source with $\rho = 0.1$.



(d) As on the left.



(e) Repetitive cheater.



(f) As on the left.

Figure 4.6: Evolution of trust values for different source types and models ($m = n = 10$).

but possible, to regain trustworthiness. A higher constant, with which the Zhao et al. 2 model increases the trustworthiness, would make it easier for a honest source to increase its trustworthiness; at the same time it would make the model less robust against repetitive cheaters. A drawback of both models is that they do not deal with error rate estimates and so have no probabilistic substantiation.

The models of Wang & Singh, Jøsang & Ismail, and ours, behave similarly for trustworthy sources and sources with $\rho = 0.1$. In the case of trustworthy sources, after a short time they all assess the trustworthiness of the source to be high. For sources with $\rho = 0.1$, all models also assign high trust-

worthiness levels that are still considerably lower than those for trustworthy sources. With a higher tolerance value ϵ, our model stays a little below the other three trustworthiness assessments. The reason is that the higher error rate makes it more probable that from time to time the tolerance interval is violated. In fact, a large threshold ϵ is suitable in scenarios where a rather large error rate is also common for trustworthy sources. In scenarios such as volunteer computing, a small threshold (e.g., $\epsilon = 1$) is more suitable since it is less tolerant towards attackers. This becomes evident in the case of a repetitive cheater. For the models of Jøsang & Ismail, a repetitive cheater needs only about 10 requests to regain his former trust value. A decrease in the aging factor λ Jøsang & Ismail's model could change how quickly a repetitive cheater could regain his former trust value. However, this would also have the effect that newer evidence has a lower weighting, which means that the cheating event itself has less impact, or in other words, is penalized less. For the model of Wang & Singh, the cheating is never forgotten, but since the cheating events are rare, the trust value still stays very high. Repetitive cheating is penalized more by our models, because after the source has cheated, this event will be weighted with a fixed weight, no matter how often the source behaves honestly afterwards. How strongly such a cheating event is penalized by our model depends on the weights; e.g., in the simulation cheating is weighted with 0.3, whereas a new positive experience gets a weight of 0.6. To reduce the probability of branding an honest source as a repetitive cheater, a higher ϵ can be used: One can see in Figure 4.6(f) that for $\epsilon = 3/12$, the lowest average trust value decreases over time; this shows that cheating attempts do not always violate the tolerance interval, however if cheating occurs several times, the probability for a violation increases.

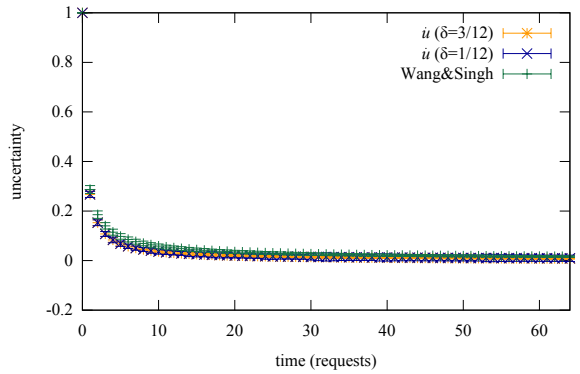
All models, except of Zhao et al. 1, are in principle able to cope with the *whitewashing* attack. In this attack, entities renew their identity to reset their trust value (see also Sect. 4.1.2.4). In the Zhao et al. 1 model, there is an incentive for whitewashing, because an entity with a new identity gets assigned the highest possible trustworthiness. For all other models, the initial trustworthiness is either zero (Zhao et al. 2), which means complete resistance to whitewashing, or 0.5, which means that whitewashing can only be effective in cases where the error rate is commonly higher than 0.5 – in all other cases, an entity with $\rho = 0.5$ will have problems in competing with other entities in the system; this is discussed in more detail later, when we examine the selection of information sources.

To conclude, in contrast to the two models of Zhao et al., our model is based on probabilistic considerations. Our model handles both honest sources and sources with higher error rates (e.g., $\rho = 0.1$) well. Furthermore, it more heavily punishes the behavior of repetitive cheaters than the model of Zhao et al. 1, and the models of Wang & Singh and Jøsang & Ismail. Finally, our model is resistant against whitewashing attacks in the same way as the models of Zhao et al. 2, Wang & Singh and Jøsang & Ismail.

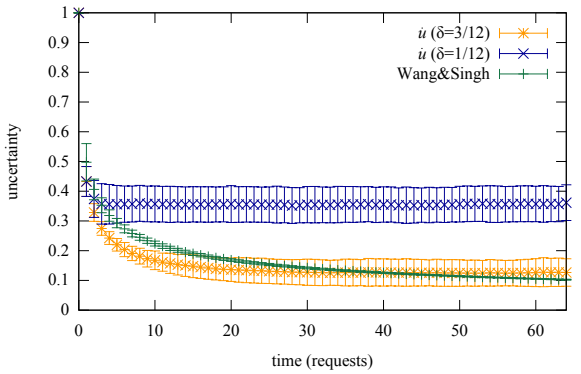
Uncertainty The uncertainty value reflects how uncertain we are about the trust value. It should decrease fast for honest information providers, and increase for unsteady sources. We prove in Appendix A.3 that our uncertainty measure fulfills the same properties as the measure of Wang & Singh [167]. In this section, we want to compare the two measures experimentally. The aim of these experiments is less to get qualitative insights on which measure is the better. We rather want to study how similarly the measures react to different source types. Wang & Singh propose computing uncertainty as:

$$\frac{1}{2}\int_0^1 \left| \frac{x^y(1-x)^{n-y}}{\int_0^1 x^y(1-x)^{n-y}dx} \right| dx \; . \tag{4.14}$$

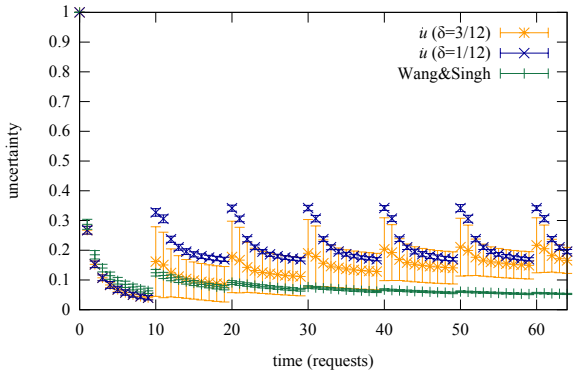
Similarly to the evaluation of the trust value in the previous section, we again consider a trustworthy source, a source with a higher error rate, but now $\rho = 0.3$, and a repetitive cheater. The simulations were run 2^{10} times, and average values and standard deviations were computed. Figure 4.7 shows the results.



(a) Trustworthy source.



(b) Source with $\rho = 0.3$.



(c) Repetitive cheater.

Figure 4.7: Evolution of uncertainty values ($m = n = 10$).

Results For a trustworthy source, the differences between Wang & Singh's measure and ours are small (Fig. 4.7(a)); still, both our uncertainty values converge slightly faster. The similarity between our measure with parameters $\epsilon = \frac{1}{12}$ and $\epsilon = \frac{3}{12}$ can be explained by the fact that an honest source violates both tolerance intervals with a negligibly small probability.

This is different for an information source with higher error rate, e.g., with an error rate $\rho = 0.3$ (Fig. 4.7(b)). There, a smaller tolerance interval ($\epsilon = \frac{1}{12}$) is violated regularly, and so the evidence cannot be combined very effectively, which results in a persistent relatively high uncertainty value; for a larger tolerance interval ($\epsilon = \frac{3}{12}$), evidence can be combined and so uncertainty decreases. The latter curve is similar to that for Wang & Singh, although our measure decreases more quickly but stays higher in the long run. Also for the larger tolerance interval, our measure will be unlikely to converge to 0, because for such a high error rate the probability is high that the tolerance interval is regularly violated; but in this case, the error rates vary a lot and the trust value is indeed less certain.

Finally, the two measures respond differently to a repetitive cheater. Our measure reacts more strongly to abrupt changes in the error rate; again, with a smaller tolerance interval more than with a bigger one. The reason is that the measure of Wang & Singh does not explicitly address the case of such abrupt changes, and as a consequence, does not apply a lower weight to the incompatible old evidence. This makes our uncertainty measure better reflect the behavior of repetitive cheaters.

To summarize, for large tolerance factors, Wang & Singh's uncertainty measure and ours react similarly to trustworthy sources and honest sources with higher error rates. If the tolerance factor is de-

Table 4.2: Avg. mean/std. dev. of error in estimating $\frac{x}{m}$ (2^7 requests, $m = n = 10$).

Type of source	$\hat{\rho}$	$1 - \dot{t}$ with W_1	$1 - \dot{t}$ with W_2	$1 - \dot{t}$ with W_3
Trustworthy	0.084/0.016	0.009/0.016	0.009/0.015	0.008 /0.014
$\rho = 0.1$	0.114/0.082	0.097 /0.067	0.097 /0.072	0.103/0.081
$\rho = 0.3$	0.154/0.114	0.133 /0.098	0.139/0.105	0.153/0.114
Repetitive cheater	0.091/0.027	0.159/0.041	0.121/0.032	0.043 /0.024
Flicker source	0.083 /0	0.332/0	0.249/0	0.083 /0

creased, our measure stays higher and so better reflects the uncertainty in case of repetitive cheaters. However, it also stays high for sources with higher error rates, which is because only the most recent evidence is considered. That means that in our model, we can learn more about a source with constant error rate when we chose a higher tolerance factor.

Jøsang & Ismail [79] propose the uncertainty measure $\frac{2}{n'_i+2}$, where n'_i is in our case $n'_i = n_i + A \cdot n'_{i-1}$ (A is again some aging factor, and $n_0 = 0$). However, this measure does not account for any changes in the number of incorrect replies, and so does not consider conflict. Besides, it does not account for sudden changes in error rate estimates. In fact, for all three kinds of sources, it would result in the same exponentially-decreasing function, and thus is not qualified for our purposes.

4.2.2.2 Trust Value as Error Rate Estimate

Provided that a source can be described by a single error rate ρ, and if looking at a single request, the best estimate of $\frac{x_i}{m_i}$ is given by $\hat{\rho}_i$ (see eq. (4.7)); this is because X and Y follow the same binomial distribution. However, as for instance in the case of a repetitive cheater, the description of the sequence of a source's error rate can be more complex than a single error rate ρ. For this reason, and because it also includes knowledge about the past, it might be better to use the trust value $(1 - \dot{t})$ as an estimator for the error rate $\frac{x_i}{m_i}$.

We compare the accuracy of the two estimators $\hat{\rho}$ and $1 - \dot{t}$ for different types of information sources. We find that for a certain choice of the weights, $1 - \dot{t}$ is in fact a better estimator than $\hat{\rho}$. As defined in Section 4.2.2.1, we consider a trustworthy source, a source with $\rho = 0.1$ and also a source with $\rho = 0.3$, and a repetitive cheater. Additionally, we consider a source that returns no incorrect replies if the request count is odd, and $m + n$ incorrect replies if it is even. We call this source *flicker source*, because the number of incorrect replies changes in the most extreme way. Flicker sources are unlikely to appear in practice, because their trust value would decrease very fast, and so they would be very "unattractive", making cheating ineffective; however, we consider them here, because they nicely show the impact of the weights on the accuracy of $1 - \dot{t}$ as estimator.

The question is, which error of estimate is lower: $|\hat{\rho}_i - \frac{x_i}{m_i}|$ or $|1 - \dot{t}_i - \frac{x_i}{m_i}|$. For the trust value computation, we considered three sets of weights, $W_1 = \{0.6, 0.3, 0.1\}$, $W_2 = \{0.9, 0.1\}$ and $W_3 = \{1\}$, reflecting different ways to discount old evidence. We ran simulations consisting of sequences of 2^7 requests, where each sequence was run 2^{12} times. For every request the mean error of an estimate and its standard deviation were computed. The average of these means and the standard deviations are shown in Table 4.2. Gray cells indicate the minimum estimation errors in each row for the respective source type.

The trust value is generally a better estimate of $\frac{x}{m}$ for honest sources, and sources with $\rho = 0.1$ or

$\rho = 0.3$. For a source with $\rho = 0.3$, weights W_1 and W_2 seem most suitable. However, for repetitive cheaters and particularly for flicker sources, these weights make the trust value a poor estimator. The reason for this is that weights W_1 and W_2 "forget" slowly, and so abrupt changes in the error rate are damped, causing the estimate to be too high or too low.

In conclusion, for all considered types of sources, $1 - \dot{t}$ with weight W_3 is a better estimator for $\frac{x}{m}$ than $\hat{\rho}$. Therefore, we use it as estimator of $\frac{x}{m}$ in the remaining part of the chapter; We write $\dot{t}^{W_3}$ as shorthand for $1 - \dot{t}$ computed with W_3.

4.2.2.3 Computational Complexity

Theorem 4.1. *For a sequence of r requests, a trust and an uncertainty value can be computed in $\mathcal{O}(r)$.*

Proof. To compute a trust value, the error rate estimates for all replies has first to be computed; this is in $\mathcal{O}(r)$. Then, the evidence of replies with similar error rates is combined, and the error rate estimates are recomputed for the combined evidences. This is again in $\mathcal{O}(r)$, because in the worst case the evidence of every pair of successive replies can be combined, and so $\frac{r}{2}$ error rate estimates have to be computed; if no evidence is combined, all the error rate estimates from the previous step can be used. In the final step, the weighted average has to be computed, which again is in $\mathcal{O}(r)$. The overall complexity is thus linear in r. For the uncertainty, instead of an error rate estimate, the standard deviation of the beta distribution is computed, which is also in $\mathcal{O}(r)$. Apart from that the procedure is the same, and thus it is in $\mathcal{O}(r)$ as well. □

4.2.3 Reducing Number of Challenges

In this section, we show how to use trust in order to reduce the number of challenges.

4.2.3.1 Approach

As humans, we feel less need to check an information source once we have a deep trust in it; this is because we believe that it is very improbable that we will receive incorrect information from it. This idea can be applied to our mechanism: The more we trust, the fewer challenges we have to use. To this end, we first combine the trust value and the corresponding uncertainty value as follows:

$$T \stackrel{\text{def}}{=} \dot{t} \cdot (1 - \dot{u}) \; . \tag{4.15}$$

This term is large only if both the trust value is high, and the uncertainty is low; in other words, if we are sure about the trustworthiness of a source. For a highly trustworthy source, we want to use a smaller number of challenges. Therefore, given some initial number of challenges n', we can determine a reduced number of challenges n as follows:

$$n = \left\lceil (1 - T) \cdot n' \right\rceil \; . \tag{4.16}$$

Because a source might become malicious at some point, it appears to be advisable always to use some minimum number of challenges $n_{\min}$. To control the speed with which trust and uncertainty reduce the number of challenges, T can be raised to a specified power, i.e., T^k; for $k > 1$ the decrease is slowed down, for $0 < k < 1$ it is accelerated. To summarize, the final number of challenges is given by:

$$n = \max\left(n_{\min}, \left\lceil (1 - T^k) \cdot n' \right\rceil\right), \text{ for some } k > 0 \; . \tag{4.17}$$

If $m + n$ is decreasing, an attacker could deduce that the number of challenges is being reduced. To counter this, the number $n + m$ should be made constant by increasing m when decreasing n.

4.2.3.2 Evaluation

To study the effects of a flexible number of challenges on the accruing costs, we compared it to the approach that uses a fixed number of challenges. A flexible number of challenges allows for more challenges to be used in the beginning to test unknown sources, reducing it once a source is known better. We conducted the experiments with $n = 16$ as starting point for a flexible n, which can be reduced down to $n_{\min} = 4$. The number of challenges is reduced according to eq. (4.17); we use $k = 2$, which makes the reduction more cautious. These values are fairly arbitrary and probably not optimal, but we show that these values can already make a flexible n superior to a fixed n under certain conditions. At this point, we do not address the issues of how to find optimal boundaries for flexible n, or optimal values of k. The tolerance factor ϵ is defined differently for flexible and fixed numbers of challenges: $\epsilon = \frac{1}{7}$ and $\epsilon = \frac{1}{10}$ respectively. We do this because for the lower numbers of challenges that occur in the flexible approach, the steps in the error rates become bigger and so a higher tolerance factor seems reasonable. The costs for an actual request i are computed as follows (compare to eq. (3.11)):

$$C_i = n_i \cdot c_c + m_i \cdot \left| \dot{t}^{W_3} - \frac{x_i}{m_i} \right| \cdot c_r \ , \tag{4.18}$$

where n_i is the number of challenges used in request i, c_c are the costs for one challenge, c_r are the costs for an error of 1 in the error rate estimate, and $\dot{t}^{W_3}$ is the trust value, but now computed with a single weight 1 (as discussed in Sect. 4.2.2.2). We fix the costs for a challenge to $c_c = 1$ and vary c_r to learn about the impact of the relation between c_c and c_r. For $c_r = 100$, this means for instance that one challenge costs as much as an error in the estimate of $1/100$. Note that the above cost function does not explicitly account for extra costs that would arise when a reply was discarded; these additional costs would for instance depend on whether the information source was paid in the case of a rejection. This means that the costs c_r also have to reflect the costs that follow from a decision that is based on an incorrect error rate estimate.

Optimal fixed number of challenges For the fixed number of challenges, we use the optimal number of challenges, as derived in Section 3.2. However, since we are only considering a mean value as estimate, and not a whole distribution, we need to adapt the error and risk functions accordingly (recall that x and y are independent here):

$$E(m, n, y, x) = m \cdot \left| \frac{y+1}{n+2} - \frac{x}{m} \right| \ , \tag{4.19}$$

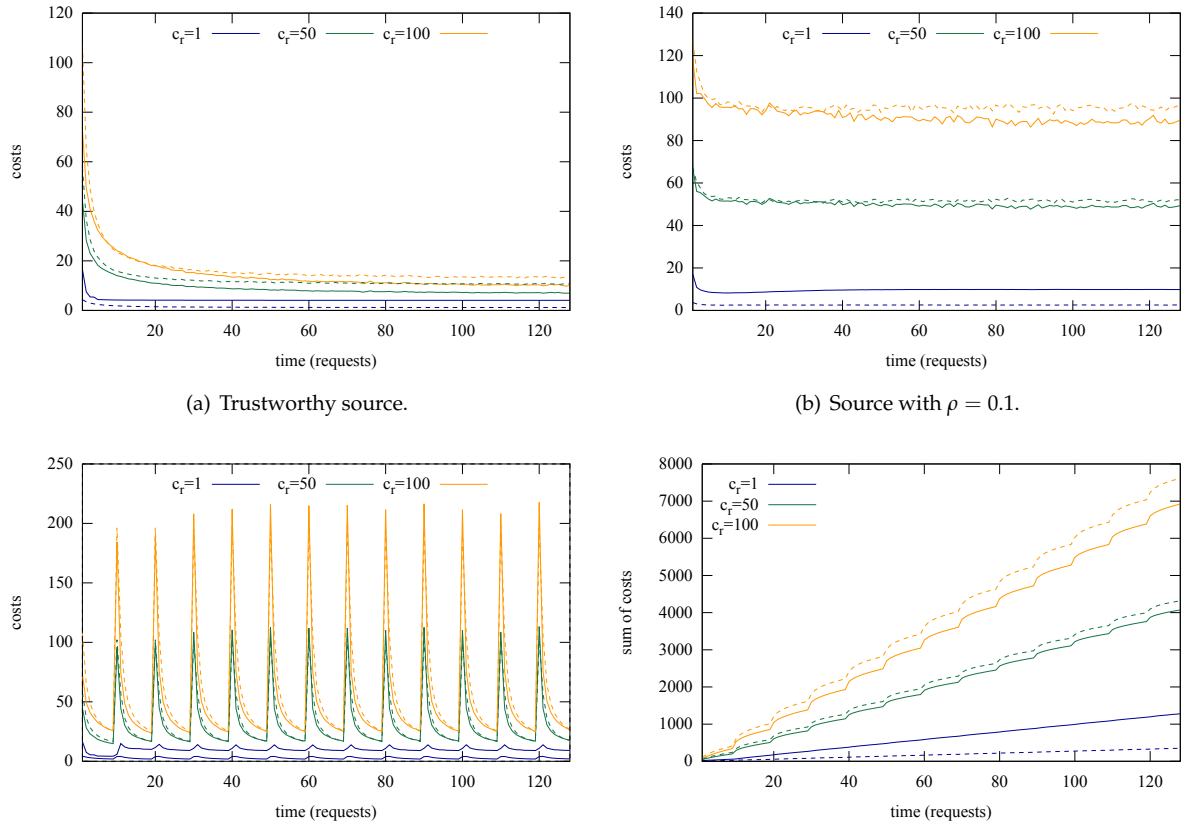
$$R(m, n) = \int_0^1 \sum_y \sum_x p(y|\rho) \cdot p(x|\rho) \cdot E(m, n, y, x) d\rho \ . \tag{4.20}$$

The optimization of eq. (3.11) with the new risk function then gives the optimal values of n as shown in Table 4.3. In the following experiments, we use the respective optimal n for each c_r.

Experiments The experiments were run 2^{12} times, and mean values computed. Results are shown in Figure 4.8. For a trustworthy source (Fig. 4.8(a)), a flexible number of challenges decreases the costs for

Table 4.3: Optimal numbers of challenges ($m = 10$, $c_c = 1$).

$c_r = 1$	$c_r = 10$	$c_r = 50$	$c_r = 100$	$c_r = 1000$
$n = 1$	$n = 3$	$n = 8$	$n = 8$	$n = 18$



(a) Trustworthy source.

(b) Source with $\rho = 0.1$.

(c) Repetitive cheater.

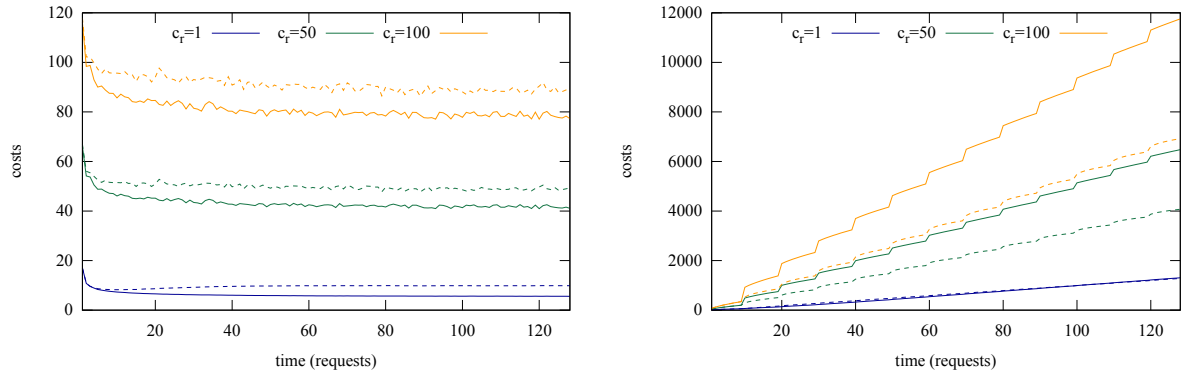
(d) Repetitive cheater (sum of costs).

Figure 4.8: Costs for flexible n (solid, $\epsilon = \frac{1}{7}$) and fixed n (dashed, $\epsilon = \frac{1}{10}$) ($m = 10$, $c_c = 1$).

$c_r = 50$, and for $c_r = 100$ after a short while; for a higher cost of the challenges c_c, a flexible n would have an even greater pay-off. If however an error in the estimate is very cheap ($c_r = 1$), the optimal fixed n is 1 and therefore a fixed n is cheaper than a flexible one. The same holds for a source with an error rate of $\rho = 0.1$ (Fig. 4.8(b)). For a repetitive cheater, it is not that evident from Figure 4.8(c) which approach is more costly. Therefore, the costs are replotted as cumulative costs in Figure 4.8(d). Then it can be seen again, that a flexible number of challenges is cheaper for higher c_r; a higher c_c would result in a better margin.

To reduce the costs in the case of $\rho = 0.1$, one can make the tolerance interval bigger. The effects of this measure are shown in Figure 4.9(a): The costs for flexible n now stay much lower, because the tolerance interval is less easily violated and so the number of challenges stays smaller; the drawback is of course that this makes the model more vulnerable to repetitive cheaters and the costs increase, as can be seen in Figure 4.9(b).

To summarize, for all considered types of sources, a flexible n pays off when the costs for a challenge



(a) Source with $\rho = 0.1$.

(b) Repetitive cheater (sum of costs).

Figure 4.9: Costs for flexible n for $\epsilon = \frac{1}{7}$ (dashed), and $\epsilon = \frac{2}{7}$ (solid) ($m = 10$, $c_c = 1$).

(c_c) are low compared to the cost of making an error in the estimate (c_r).

4.2.4 Trust-Based Selection of Sources

At this point, we know how to compute trust and uncertainty values and how to use them to reduce the overhead produced by the challenges. It remains to be shown, how these values can be used in the selection of future interaction partners. This decision is an important step in scenarios such as web service selection, worker selection in volunteer computing, etc.

4.2.4.1 Approach

The main problem that we face here is the *explore vs. exploit* dilemma described in Section 4.1.2.4. One approach to this problem is "Boltzmann exploration" [113]. In this approach, a certain decision d is chosen with a probability that is proportional to the estimated value of the decision V_d:

$$p_B(d) = \frac{e^{B \cdot V_d}}{\sum_i e^{B \cdot V_{d_i}}}, \tag{4.21}$$

where d is the decision under consideration, V_{d_i} is the estimated value of decision d_i, and $B \geq 0$ is a parameter that we call the "Boltzmann parameter" (usually called the "temperature"). The greater the Boltzmann parameter is, the more impact the value of the decision has on the probability p_B. As B approaches zero, the probabilities of different decisions become more and more uniform. Initially, when the expected values of a decision are still very uncertain, a low B, which can then be increased over time, is reasonable.

The Boltzmann exploration would be suitable for our purposes if we had an estimate for how good the choice of a certain information source is. Fortunately, the trust in a source accounts both for his attitude and his competence, and thus is a good indicator of the value of choosing this source. Let T_a denote the value that combines trust value and uncertainty for an information provider a, as defined by eq. (4.15). Then, the next request is made to an information provider a with the following probability:

$$p_B(a) = \frac{e^{B \cdot T_a}}{\sum_i e^{B \cdot T_i}}. \tag{4.22}$$

Note that the combined value T_a does not distinguish between the two cases of (1) high trust but also high uncertainty, and (2) low trust but low uncertainty. The source in the second case is strongly believed to be untrustworthy, and so, it should be selected with a lower probability than the source in the first case, as long as the system explores a lot. A way to account for this would be to compute the combined trust value T by weighting the uncertainty with a function of the Boltzmann parameter (compare to eq. (4.15)):

$$T = \dot{t} \cdot (1 - \frac{B}{B_{\max}} \cdot \dot{u}) \ , \tag{4.23}$$

where $B_{\max}$ is the maximal Boltzmann parameter used by the system. This extension reduced the impact of the uncertainty on the combined trust value as long as the Boltzmann parameter is low, i.e., the system explores a lot. In our simulations, we do not consider this extension and leave its evaluation subject to further work.

Multiple Selection In a scenario like volunteer computing, not just a single worker is selected at a time, but a set of workers. This can be done by applying the Boltzmann exploration to the set of remaining workers after each selection. A more efficient but equivalent approach is to successively select workers with probability as above, while ignoring multiple selections of the same worker, until enough workers have been selected.

Blacklisting The idea of *blacklisting* is to mark sources that are thought to be untrustworthy, and never to consider them again in the future. The danger of this is that honest information sources might be blacklisted erroneously. To avoid this problem, we propose to mark them only if both the trust value and the uncertainty are low enough. This is reasonable, because a low uncertainty value signifies that enough evidence has been amassed to back the trust value. This would certainly reduce the number of false positives, i.e., the number of blacklisted honest sources. On the other hand, it would probably increase the number of false negatives – the untrustworthy sources that are not blacklisted. To determine good thresholds for this decision is outside the scope of the thesis, but would be an interesting direction for further research.

4.2.4.2 Evaluation

The trust model is now equipped with the capability of selecting one out of a set of potential information sources. This introduces more dynamics into the application of the model and makes its evaluation much more complex compared to the evaluation against a fixed strategy as in Section 4.2.2.1. In the following, we first discuss the effectiveness of a whitewashing attack. Then, we show by means of simulation the impact of different settings of source types and parameters on the performance of the trust model with selection. The objective is that in the long run the model will tend to select trustworthy sources only.

Whitewashing Attacks Whitewashing attacks are effective if a source can become more attractive by changing its identity. Our initial trust value of 0.5 gives an incentive to renew an identity only if the trustworthiness has fallen below 0.5. Now, the higher the Boltzmann parameter B is set, and the more sources with higher trust values are already known, the lower is the probability that a source with a trust value of 0.5 will be reconsidered. Therefore, *whitewashing attacks* are especially ineffective in scenarios where the usual error rate is considerably lower than 0.5; in volunteer computing it is for instance

typically below 0.0355 [88]. Then, at some point a high number of sources with a higher trust value will be known, making it hard for newcomer information sources to gain attention from a trustor. If B is very low, the selection procedure takes less account of the trust value; however, there is then also no incentive to reset it to 0.5.

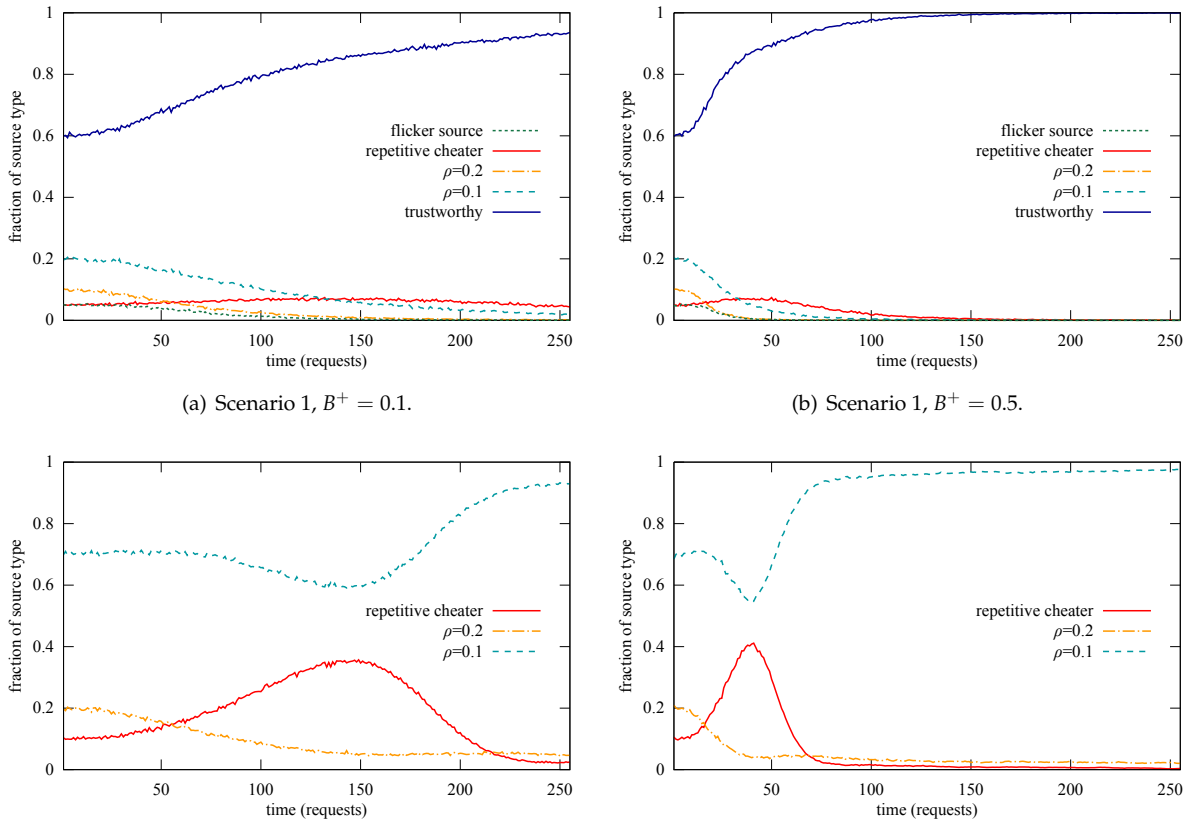
Impact of Sybil Attack By creating many fake identities, an attacker can increase the probability that these are selected in the phase when the Boltzmann exploration tends to explore. These false identities can provide information with relatively high error rates, and yet, because they are numerous, the probability that some of them will be selected in a more exploit-oriented phase is high. This is later confirmed by our experimental findings for the tuning of the trust model (Sect. 4.3.3.4). There we found that the higher the fraction of attackers, the more aggressively can they define their attack strategies, because the less they have to compete with honest sources. This implies that a high probability of a Sybil attack should be countered by a longer exploration phase.

Experiments We ran simulations where the requester used the Boltzmann exploration approach to select an information source. The Boltzmann parameter was initially set to 1 and incremented by 0.1 or 0.5 after each request; this increase is denoted by B^+. We considered two different scenarios (see Sect. 4.2.2.1 and Sect. 4.2.2.2 for the descriptions of the source types):

Scenario 1 This scenario aims to show how quickly trustworthy sources can be recognized. To this end, the population of sources consisted of a majority of trustworthy sources (60%), and a mixture of incompetent or malicious sources: 20% sources with error rate $\rho = 0.1$, 10% with $\rho = 0.2$, 5% repetitive cheaters and 5% flicker sources.

Scenario 2 This scenario describes the case where an error rate of $\rho = 0.1$ is common, and thus it becomes harder to identify malicious sources. A majority of sources had error rate $\rho = 0.1$ (70%); a smaller fraction had a higher error rate of $\rho = 0.2$ (20%), and 10% were repetitive cheaters. We did not consider flicker sources, because the first scenario had already revealed that these are easily recognized and ignored. Note that repetitive cheaters have an intermediate error rate of just $\rho = 0.00221$, which requires that they are more powerful than the most competent honest sources in this scenario, making it a challenging scenario for the trust model.

The simulations were repeated 2^{13} times and average values were computed to give statistically significant results. The results are shown in Figure 4.10; the curves show the average of how often the respective source type has been selected at each point in time. In scenario 1, the model selects more and more trustworthy sources over time. While the sources with error rates $\rho = 0.1$ and $\rho = 0.5$ were filtered out at all times, the repetitive cheaters were able to increase their importance for a certain time. This is because the repetitive cheater first increases its trust value by pretending to be trustworthy, and so delays being perceived untrustworthy. Still, at some point, it will be sidelined. For $B^+ = 0.5$, the sidelining of untrustworthy sources happens earlier. This can be explained by the fact that the trust value increases in importance more quickly for the selection. Similarly in scenario 2, the repetitive cheaters succeed in increasing their frequency of selection for a certain time. This works better than in scenario 1, because for the first ten interactions, their error rate is much lower than that of their best competitors ($0.00221 \ll 0.1$). However, at some point, which again is earlier in time for $B^+ = 0.5$, the trust value of most repetitive cheaters has dropped and so other sources start to regain importance, and finally to dominate repetitive cheaters.



(a) Scenario 1, $B^+ = 0.1$.

(b) Scenario 1, $B^+ = 0.5$.

(c) Scenario 2, $B^+ = 0.1$.

(d) Scenario 2, $B^+ = 0.5$.

Figure 4.10: Fractions of selected source types ($m = n = 10$, $\epsilon = \frac{1}{12}$, $(w_i)_i = (1, 3, 6)$).

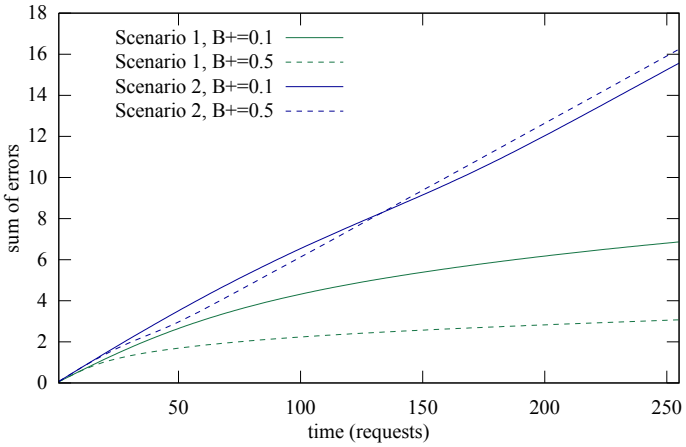


Figure 4.11: Errors in error estimates $\dot{\imath}^{W_3}$ (sum).

The cumulative error of the estimate $\dot{\imath}^{W_3}$ is shown in Figure 4.11. For scenario 1, the increase in the sum of estimation errors decreases less over time. This is due to the cumulative fading in of trustworthy sources, and fading out of other sources. For scenario 2, the sum of errors evidently lies above the sum of errors for scenario 1, because there are no trustworthy sources with a very low error rate; this can

be seen in Table 4.2 on page 58. A similar reason explains why a higher B^+ causes higher errors in the long run: The estimation error for sources with $\rho = 0.1$ is higher than that for repetitive cheaters (again, see Table 4.2), and repetitive cheaters are earlier faded out for higher B^+. In fact, this shows that in this scenario a higher tolerance factor ϵ would have been wise, because it would allow the trust model to accumulate more evidence for sources with higher error rates, and thus produce more accurate error rate estimates for them.

To summarize, the trust model is capable of increasingly selecting trustworthy sources. How fast the trust model fades out untrustworthy sources depends on several factors; amongst others on the composition of source types and in particular on the trust model parameters such as the Boltzmann parameter or the tolerance factor. However, given that it is not clear what strategies malicious sources will use, it is hard to know what an optimal parameter setting of the trust model is. Therefore, we propose in the following a procedure that aims to find optimal parameter settings for evidence-based trust models.

4.3 Tuning Evidence-Based Trust Models

In the literature, it is often left open how systematically to find suitable configurations for the proposed trust models. The effectiveness of a proposed trust model is usually evaluated for specific parameter settings (configurations) of the model. Although such an evaluation can show how well the model works for these specific configurations, the evaluation does not tell us whether the model could perform better if the configuration were improved. Consequently, methods for automatically optimizing the parameters of trust models are necessary. The problem can be formulated as finding a configuration for the trust model, from a finite set of possible configurations that yields the highest expected payoff. In the following, we present a generic approach to this problem.

Our Approach The idea presented here is to consider the deployment of a trust model as a *game* (for an introduction to game theory see [53]). By applying a game-theoretic solution concept, one can identify the configuration of the trust model which yields the highest expected utility to the trustor. However, this requires knowing the strategy a rational attacker is most likely to use against a given trust model configuration. In order to search for such strategies, we use a *genetic algorithm* [63]. To provide a proof of concept, we apply this approach to the trust-model presented in Section 4.2.

4.3.1 Trust Model Deployment as Game

We model the deployment of a trust model as a *normal-form game* of the trustor against a set of opponents, the potential trustees. An opponent is a service provider that is either trustworthy, *or* malicious, in which case we will refer to it as an "attacker". Let such a game consist of a sequence of rounds. In each round, the trustor selects an opponent and requests some service τ. If the selected opponent is trustworthy, he acts in a predefined way (in our case for instance as defined in Sect. 4.2.2.1). If the opponent is an attacker, he can respond with some action ψ chosen from a possibly infinite set of actions Ψ.

Let the finite set S_T contain all possible configurations of the trust model under consideration;[6] a strategy $s_T \in S_T$ determines for any given game history how the trustor decides which service τ to request from which opponent in the next round. Analogously, let the possibly infinite strategy space

[6]To ensure that S_T is finite, parameters of the trust model have to be bound if necessary, and continuous parameters have to be discretized.

Table 4.4: Payoff matrix for an entire game.

		attack strategies		
		s_{A1}	s_{A2}	...
trust model configurations	s_{T1}	$\mathcal{U}_T(s_{T1}, s_{A1})$, $\mathcal{U}_A(s_{T1}, s_{A1})$	$\mathcal{U}_T(s_{T1}, s_{A2})$, $\mathcal{U}_A(s_{T1}, s_{A2})$	...
	⋮	⋮	⋮	...
	s_{Tn}	$\mathcal{U}_T(s_{Tn}, s_{A1})$, $\mathcal{U}_A(s_{Tn}, s_{A1})$	$\mathcal{U}_T(s_{Tn}, s_{A2})$, $\mathcal{U}_A(s_{Tn}, s_{A2})$	...

S_A of an attacker contain all attack strategies that can be deployed against the considered trust model; a strategy $s_A \in S_A$ describes which action ψ an attacker will choose for a given game history and a requested service τ. We also call s_A an *attack strategy*.

For each pair (τ, ψ), two functions $\mathtt{U}_T$ and $\mathtt{U}_A$ define, for the trustor and the attacker respectively, the real-valued utilities for a single round in which the trustor requests τ, and the opponent responds with ψ. Another utility function $\mathcal{U}_T : S_T \times S_A \mapsto \mathbb{R}$ defines the expected utility of the trustor for an entire game. Analogously, a function $\mathcal{U}_A : S_T \times S_A \mapsto \mathbb{R}$ defines the expected utility of an attacker for an entire game. Table 4.4 illustrates the payoff matrix containing the expected utilities for an entire game. Rows represent strategies of the trustor (trust model configurations), and columns represent attack strategies. We write (S_T, S_A)-*game* for a game where a trustor and an attacker can choose from strategy spaces S_T and S_A respectively. Analogously, we write (s_T, s_A)-*game* for a game, where a trustor and an attacker play strategies s_T and s_A respectively.

4.3.1.1 Solution Concept

If there is a procedure that finds an optimal trust model configuration, then a sophisticated attacker could run this procedure and thus would know the optimal trust model configuration. Also to be in accordance with *Kerckhoff's principle* [83], we should therefore assume that an attacker knows the trust model configuration that will be used.

Hence, we can analyze an (S_T, S_A)-game in the form of a "Stackelberg competition" (e.g., see [53, pp. 67–69]). In a Stackelberg competition, the first player, the "leader", chooses his strategy first. The second player, the "follower", can then choose his strategy with the knowledge of the leader's strategy. In our case, the trustor represents the leader, who has to choose a trust model configuration $s_T \in S_T$ as his strategy. The attacker takes the role of the follower and can choose, knowing the trust model configuration s_T, a strategy $s_A \in S_A$ that produces the most favorable outcome for him; such an optimal attack strategy is called a "best response" to the trust model configuration s_T.

Stackelberg competitions originally address the situation where two firms compete by varying output levels, and so the payoffs of leader and follower are usually linked linearly. In contrast, in our case, we consider all kinds of payoff matrices. As we will see, a similar reasoning to that of the original Stackelberg competition is possible.

4.3.1.2 The Stackelberg* Outcome

In what follows, we assume *perfect rationality* of the players: They know how to maximize their utilities and act accordingly. Since there might be several best responses to a trust model configuration, we

assume the worst case for the trustor: The attacker chooses from the set of best responses the one that minimizes the expected utility of the trustor. This is reflected by the following notation.

Notation 4.1. *For an (S_T, S_A)-game and a trust model configuration $s_T \in S_T$, we use* $\text{br}^*(s_T) \in S_A$ *to denote a best response to strategy s_T that causes the worst utility for the trustor among all best responses to s_T (if there are several). Formally, for any s_T and* $\text{br}^*(s_T)$ *it holds:*

$$\forall s_A \in S_A : \left[\mathcal{U}_A(s_T, s_A) \leq \mathcal{U}_A(s_T, \text{br}^*(s_T))\right] \tag{4.24}$$

$$\wedge \left[\mathcal{U}_A(s_T, s_A) = \mathcal{U}_A(s_T, \text{br}^*(s_T)) \right. \tag{4.25}$$

$$\left. \Rightarrow \mathcal{U}_T(s_T, s_A) \geq \mathcal{U}_T(s_T, \text{br}^*(s_T))\right] . \tag{4.26}$$

Using this notation, we define a solution concept as follows:

Definition 4.2 (Stackelberg* outcome)**.** *Assuming a rational attacker in an (S_T, S_A)-game, strategy pair $(s_T, \text{br}^*(s_T))$ is a* Stackelberg* outcome *if there is no other strategy $s'_T \in S_T$ for which the following holds:*

$$\mathcal{U}_T(s'_T, \text{br}^*(s'_T)) > \mathcal{U}_T(s_T, \text{br}^*(s_T)) . \tag{4.27}$$

According to this solution concept, a leader will choose a configuration s_T that maximizes his utility knowing that a rational follower's strategy is to play $\text{br}^*(s_T)$. This solution concept actually corresponds with the *Stackelberg outcome* [53], apart from the fact that the former requires the follower to minimize the leader's utility in the second step.

Theorem 4.2. *A Stackelberg* outcome constitutes a Nash equilibrium.*

To prove Theorem 4.2, we have to show that for a Stackelberg* outcome, players cannot get greater utilities by unilaterally changing their strategy.

Proof. Let $(s_T, \text{br}^*(s_T)) \in S_T \times S_A$ be a Stackelberg* outcome. The trustor will not change his strategy to some different $s'_T \in S_T$ because he knows that the attacker would then play $\text{br}^*(s'_T)$ which would not cause a higher utility for the trustor – because following Def. 4.2 we have:

$$\mathcal{U}_T(s_T, \text{br}^*(s_T)) \geq \mathcal{U}_T(s'_T, \text{br}^*(s'_T)) .$$

The attacker also has no incentive to play a different strategy $s_A \in S_A$ with $s_A \neq \text{br}^*(s_T)$, because his utility could not increase (see eq. (4.24)). □

4.3.2 Tuning Procedure

To find an optimal trust model configuration, we have to find a Stackelberg* outcome. This is because it maximizes the trustor's expected utility under the assumption of a rational attacker. Consider a restricted payoff matrix as shown in Table 4.5. As before, rows represent the strategies of the trustor, i.e., trust model configurations. The only column now represents the attacker's best responses br* to the corresponding trust model configuration. Every row, in which the expected utility for the trustor is at least as high as in all other rows, provides a Stackelberg* outcome. So, we choose the trust model configuration from such a row. If there are several such rows, we randomly pick one.

Table 4.5: Restricted payoff matrix for an entire game.

		best response $\text{br}^*(s_{Ti})$
trust model configurations	s_{T1}	$\mathcal{U}_T(s_{T1}, \text{br}^*(s_{T1}))$, $\mathcal{U}_A(s_{T1}, \text{br}^*(s_{T1}))$
	$\vdots$	$\vdots$
	s_{Tn}	$\mathcal{U}_T(s_{Tn}, \text{br}^*(s_{Tn}))$, $\mathcal{U}_A(s_{Tn}, \text{br}^*(s_{Tn}))$

4.3.2.1 Finding Best Responses

To construct the restricted payoff matrix as illustrated in Table 4.5, we have to know the attacker's best response to each trust model configuration. This can be done by searching for it in the search space consisting of possible attack strategies.

Attack Strategy Search Space Let Ψ be the set of all possible actions ψ an attacker can take in a single interaction. Then the full attack strategy space for a game with r rounds is given by Ψ^r. For a reasonable number of rounds (e.g., $r = 50$), this search space is already vast, even if an attacker can only take two different actions. To reduce the size of the search space, we take another approach: We construct the search space from a set of *basic strategies*, each of which is parameterized with a finite set of possibly continuous parameters. In Section 4.3.3.2, we define these basic attack strategies for a concrete application scenario. The following example illustrates how a basic attack strategy can be described:

Example 4.2. *Let $\rho \in [0, 1]$ be a continuous and $i \in \mathbb{N}$ be a discrete parameter; in every i*th *round, the attacker returns incorrect information with error rate ρ, and otherwise behaves in a trustworthy manner.*

The drawback of this approach is that one might neglect important basic attack strategies. In other words, the attack strategy search space is not complete. As a consequence, the parameter settings selected by our tuning procedure are not guaranteed to be optimal (even if a search algorithm that guarantees finding a global optimum is used). However, we clearly need to restrict the search space, because a search would become infeasible otherwise. The more comprehensive the set of strategies is, and the more sophisticated the strategies are, the more likely it contains the actual best response. Additionally, by adding only a selection of attacks to the search space, a trust model can be optimized specifically against these attacks. In any case, the resulting space can still be very large, and so an appropriate search algorithm has to be selected.

Search Algorithm Because of the size of the search space, an *exhaustive search* or *uninformed search* [129] is not suitable. Instead, we propose to use a genetic algorithm, because it can be flexibly adapted to very different search spaces. Actually, depending on the attack strategy space, the fitness function can be nonlinear and even partially non-continuous – if this is taken to extremes though, even a genetic algorithm will struggle. Furthermore, the structure of genetic algorithms makes them good candidates for parallelization. And finally, they can handle continuous parameters in the attack strategies. However, there are also arguments against genetic algorithms in our case. Because a genetic algorithm is a *local search method* it does not guarantee convergence to global optima. So, the performance of the algorithm

depends to a certain extent on its configuration. However, the results that we describe later provide evidence that we succeeded in tuning the genetic algorithm sufficiently.

The Genetic Algorithm For our purposes, we specify the different components of the genetic algorithm[7] as follows:

Individuals The individuals represent the attackers. An individual is thus defined by a specific basic attack strategy parameterized in a certain way. At the beginning of the algorithm each individual is initialized with a random basic attack strategy with random parameters.

Fitness Function The fitness of an individual using a certain attack strategy s_A (against a trust model with configuration s_T) is given by the utility function $\mathcal{U}_A(s_T, s_A)$. The value of the utility function is determined by averaging over the utilities obtained in a statistically significant number of simulated (s_T, s_A)-games.

Selection An individual is selected for reproduction with a probability that is proportional to the individual's fitness. Parameter $\mathcal{S}$ defines the *selection pressure*, which specifies the impact of differences in the fitness on the resulting selection probabilities. To be more specific, if I is the population of individuals and $f(i)$ is the fitness of individual $i \in I$, then we select i for reproduction by means of Boltzmann exploration (see also Sect. 4.2.4):

$$\frac{e^{\mathcal{S}f(i)}}{\sum_{j \in I} e^{\mathcal{S}f(j)}} \,. \tag{4.28}$$

Reproduction Given that two individuals have been selected for reproduction, a new individual is created that takes a part of the parameters from the "mother" and the other part from the "father". This affects the parameters of the attack strategy as well as the basic attack strategy used. How this is done in a reasonable way depends on the form of the attack strategies. In our algorithm, mother and father are not necessarily part of the subsequent generation. However, a certain number of individuals with the highest fitness is always passed on to the next generation; this is called "elitism" and ensures that good solutions are not lost. The size of the population stays constant.

Mutation For a certain fraction of the individuals that result from reproduction, one or more parameters and/or the basic attack strategy are *mutated*, i.e., changed in a defined way. To a continuous parameter one can for instance add a random number following a Gaussian distribution. Again, this depends on the form of the attack strategies.

Termination The algorithm terminates if either of the following happens:

- the fitness of the best individuals of a certain number of consecutive generations does not change significantly,
- a certain number of generations has been reached. In our case, where the fitness function is computed by means of simulation, this suggests that the average utilities are fluctuating too much. As a countermeasure the number of games averaged over should be increased.

[7] See Section 2.3.8 for a quick introduction to genetic algorithms, and helpful references to the literature.

Performance of Genetic Algorithm The performance of the genetic algorithm proposed above suffers from the expensive computation of the fitness function. This is due to the fact that a game has to be repeated often enough to make the results statistically significant. Nevertheless, genetic algorithms are well-suited to parallel execution, which can speed up the overall computation considerably. Also, the set of trust model configurations can be pruned by removing implausible settings. Alternatively, the procedure can be run several times, starting with a small set of considered trust model parameter settings, and iteratively making it more fine-grained in promising areas.

4.3.2.2 On the Applicability of Game-Theory

In our case, the strategies of the trustor are concrete trust model configurations (see Sect. 4.3.1). Thus, our solution concept yields an optimized but fixed trust model configuration. In game-theoretic terms this means that the trustor basically plays a "pure strategy". We believe that to compute the payoff for a "mixed strategy", i.e., a strategy where the trustor uses different configurations with certain probabilities, would hardly be feasible in practice. The reason for this would be the complex dynamics in a game, e.g., caused by the honest players and the dynamic adaptation of the attackers to the changes in the trust model configuration. This raises the question of whether game theory is the right approach for handling this complexity. We rather suggest that a configuration determined by a procedure like ours serves as a starting configuration, and that the trust model should be equipped with capabilities to adapt itself at runtime according to feedback it gets from its environment.

4.3.3 Application

We apply the tuning procedure to an exemplary volunteer computing scenario in a round-based manner. In this scenario, a master assigns work units to workers, which are expected to return the correct result. Because workers may return incorrect results, the master wants to use the trust model with selection presented in this chapter. The master will accept a worker's reply if and only if the error rate is estimated to be below a threshold e set to 0.1 (the procedure could also be run for any other threshold). After having been assigned a work unit, a worker is busy for a certain time and cannot be selected for several rounds. For our simulations, we set this number of rounds to a small value of 10, so that in our simulations consisting of 2^8 rounds, the same malicious worker could be selected many times; this means that the utility of an attacker is determined more by its strategy than by chance, which results in a more efficient and reliable optimization of the attack strategies.

The trust model can be parameterized by the increase in the Boltzmann parameter B, tolerance factor ϵ, and the weights w_i (see Sect. 4.2.4 and Sect. 4.2.1.2). The more parameters of the trust model and the more fractions f_a are considered, the more precisely can the final expected utility of the master be computed. To keep the computational efforts reasonable in our case, we consider the following subset of possible trust model parameters:

- $B^+ \in \{0, 0.05, 0.1, 0.25, 0.5, 0.75, 1, 1.5, 2.5, 5\}$
- $\epsilon \in \{\frac{1}{10}, \frac{1}{4}, \frac{1}{2}\}$
- $W_1 = (1), W_2 = (0.8, 0.2), W_3 = (0.7, 0.2, 0.1)$

For the simulations, we varied the fraction of workers with malicious intent, denoted by f_a. We ran the simulation with four different fractions f_a, namely 0, 0.1, 0.5 and 0.9. The final expected utility of the master was then computed as a weighted average over the expected utilities for different f_a.

This implicitly assumes that attackers know the fraction of attackers f_a in order to choose their best response, because we searched for their best attack strategy for the different fractions independently. Alternatively, one could vary f_a during the execution of the genetic algorithm, in order to optimize the attackers' strategies for all possible fractions f_a. However, the fitness function would then change for every population, possibly in contrary ways, making it much harder for the genetic algorithm to converge. Therefore, we chose the first approach.

What still needs to be defined at this point are the utility function of the master, the utility function of the malicious workers, and their basic attack strategies.

4.3.3.1 Utility Functions

For both attacker and master, we have to define their utility for a single round, in which the master requests τ and the worker takes action ψ.

Attacker If the reply of an attacker is accepted, he gets credit for $m + n$ work units; additionally, he has saved resources by returning $x + y$ faked results, which provide utility as well. If the reply is rejected, the attacker has unnecessarily computed $m - x$ plus $n - y$ work units, which results in the negative utility of $-(m - x) - (n - y) = x - m + y - n$. These two cases define the utility function for the attacker:

$$\mathtt{U}_A(\tau, \psi) = \begin{cases} m + x + n + y, & \text{if } \dot{t}^{W_3} \leq e \\ x - m + y - n & \text{if } \dot{t}^{W_3} > e \end{cases} \tag{4.29}$$

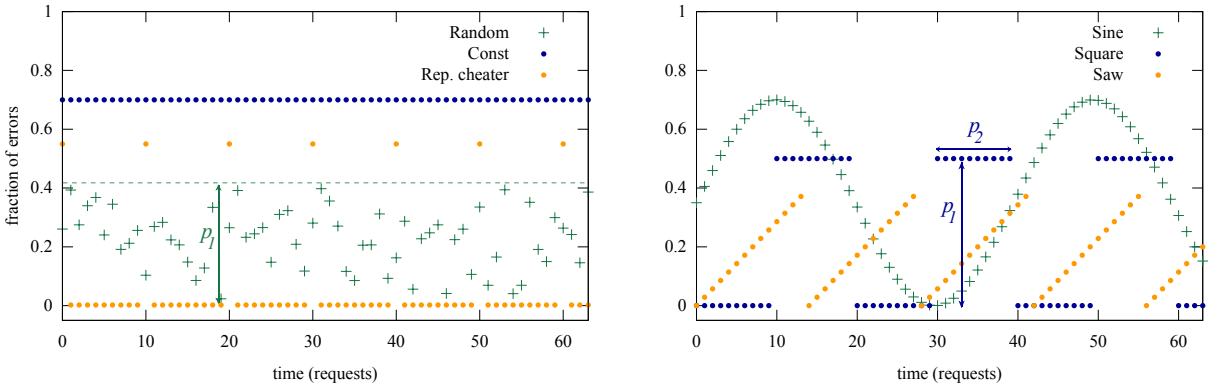
Master For the master, there are three different cases:

$$\mathtt{U}_T(\tau, \psi) = \begin{cases} m - x, & \text{if } \dot{t}^{W_3} \leq e \text{ and } \frac{x}{m} \leq e \\ m - x - (|e - \frac{x}{m}| \cdot m) & \text{if } \dot{t}^{W_3} \leq e \text{ and } \frac{x}{m} > e \\ -m, & \text{if } \dot{t}^{W_3} > e \end{cases} \tag{4.30}$$

In the first case, the master accepts the reply, and the reply's error rate is indeed smaller than e: We define the master's utility by the number of correct results in M. If the master accepts the reply although its error rate is higher than e, we additionally subtract the number of errors above the threshold. In the case that he has to reject the request, we define the utility to be the negative number of real requests, because these have to be requested again. Note that if only correct results are returned to the master within a game, that is, in the best case, its average utility per round is m. The different terms could also be weighted with different cost parameters, e.g., the error in the second case could be more costly in practice than the number of lost replies in the third case. However, since we have no concrete reasons to do otherwise, we set all costs to 1.

4.3.3.2 Attack Strategies

The set of actions Ψ of an attacker, consists of all possible numbers of incorrect results to a given number of single requests. More precisely, if a request consists of $m + n$ single requests, the worker can decide to return one to $m + n$ incorrect requests; this can also be expressed by the fraction of incorrect results within the total number of results. Attack strategies can then be described by sequences of these fractions. We modeled six basic attack strategies, each of which was parameterized with up to 2 parameters $p_1 \in [0, 1]$ and $p_2 \in \mathbb{N}$. Figure 4.12 illustrates the attack strategies with different parameters for each,



(a) Random, Constant and Repetitive cheater.

(b) Sine, Square and Saw.

Figure 4.12: Attack strategies that form the attack strategy search space.

and illustrates for two of the strategies the effect of the parameter(s). The attack strategies are defined as follows:

Random For each interaction draw the fraction of errors randomly from a uniform distribution over the interval $[0, p_1]$.

Constant Always return a fraction of errors of p_1.

Repetitive cheater (As defined in Sect. 4.2.2.1) Repeatedly pretend to be a honest worker for a number of p_2 rounds, then return a response with error rate p_1.

Sine For the ith interaction, choose the following fraction of errors: $p_1 \cdot (0.5 + 0.5 \cdot \sin_{p_2}(i))$, where p_2 parameterizes the frequency of the sine function. The 0.5-constants normalize the sine function into $[0, 1]$.

Square Choose a fraction of errors similar to Sine: $p_1 \cdot (0.5 - 0.5 \cdot \text{sgn}(\sin_{p_2}(i)))$. For the first interaction pretend to be honest.

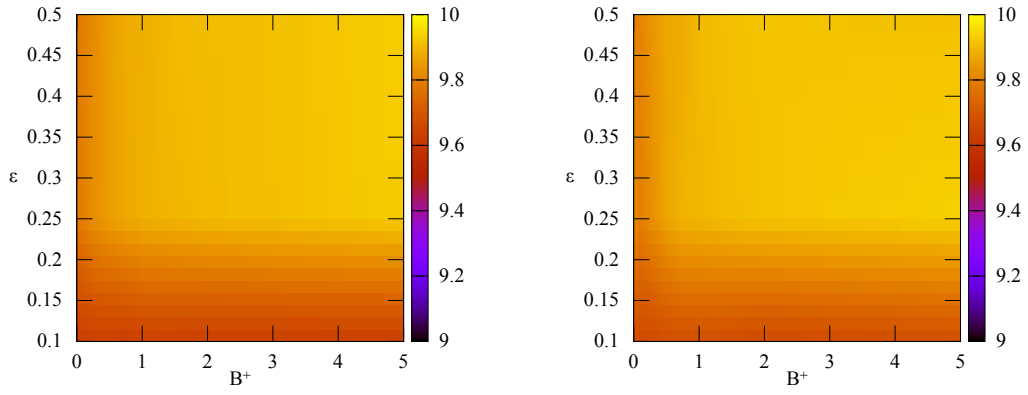
Saw For the ith interaction, choose fraction of errors $p_1 \cdot (\frac{i \bmod p_2}{p_2})$.

We modeled the basic strategies $s \in \{1, \ldots, 6\}$ as another parameter that the attacker had to choose. So, the genetic algorithm optimized the pair (s, p_1, p_2) in order to maximize the attacker's utility against a given trust model configuration.

4.3.3.3 Genetic Algorithm Implementation

To have a reasonably high significance, the fitness function was computed by taking the average utilities of master and attackers over 2^9 games, each consisting of 2^8 rounds. The population consisted of 2^7 individuals, which, with a selection pressure of $\mathcal{S} = 20$ and a number of 13 mutations, resulted in a fast convergence. The search was terminated if either the fitness ceased to improve by an amount of at least 0.005 for a sequence of four rounds, or if a maximum of 2^4 rounds was reached.

An individual represented a master with a certain trust model configuration on the one side, and a set of workers on the other side. The number of workers was kept small (50) to ensure a high number of interactions within a relatively small number of rounds. A certain ratio of workers was given a malicious



(a) $f_a = 0$. (b) $f_a = 0.1$.

Figure 4.13: Expected utilities of the master against optimized attack strategies (W_1).

intent, and all of them followed the same attack strategy, which was changed through mutation and reproduction with other individuals.

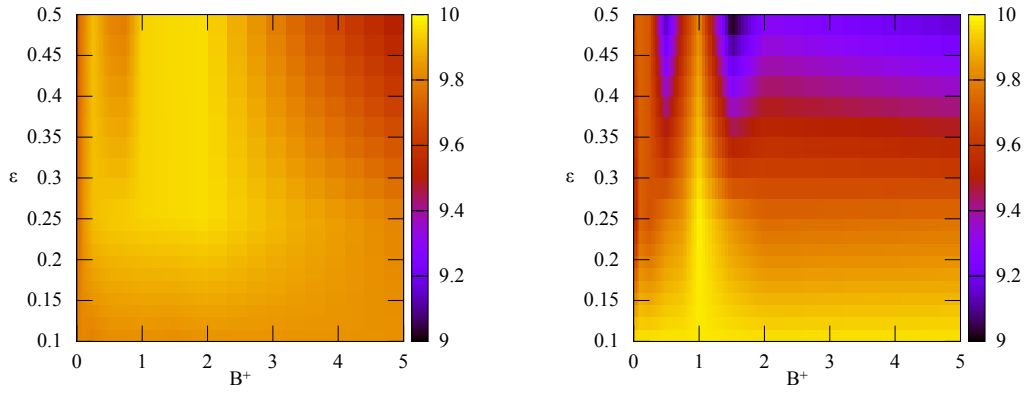
Reproduction Reproduction is implemented by selecting two individuals and creating a new one, repeatedly until enough individuals are produced. With a probability of 0.2 parameters p_1 and p_2 were taken from the same parent, and from different parents otherwise. The attack strategy s is selected with probability 0.75 from the parent with the higher fitness, and from the other parent otherwise. Elitism is implemented by passing the two best individuals on to the next generation without modifying them.

Mutation Mutation changes one parameter at a time: parameter p_1 and p_2 with probability 0.4 each, and attack strategy s otherwise. To p_1 or p_2 a value is added that is taken from a Gaussian distribution with mean 0 and standard deviation 0.05; if the value falls outside of $[0, 1]$, it is chosen from a uniform distribution over $[0, 1]$. If mutation changes the attack strategy, this is randomly picked from $\{1, \ldots, 6\}$.

4.3.3.4 Results

The results are shown in Figures 4.13 and 4.14; the results are interpolated with ten values between each pair of measurements. The plots show the expected utilities $\mathcal{U}_T$ of the master, divided by the number of rounds; since we are using $m = 10$, the highest reachable expected utility is $\mathcal{U}_T = 10$. In each plot, two of the three considered trust model parameters are presented: The x-axis gives increase of the Boltzmann parameter (B^+), and the y-axis the tolerance factor (ϵ). The different weights W_1, W_2 and W_3 led to similar results, and are therefore relegated to Appendix D.

Lower Fractions of Attackers Figure 4.13 shows that for a low fraction of attackers, the utility is generally as high as if there are no attackers at all. These results suggest that malicious workers are forced to return mainly correct results, because otherwise they could not compete with the trustworthy workers. Actually, for certain parameter settings the utilities of the master are even better in the scenarios with few malicious workers; where this is the case can be seen better in Appendix D, where a table lists numeric values for all the utilities of the master. The reason for this is that when there are only



(a) $f_a = 0.5$. (b) $f_a = 0.9$.

Figure 4.14: Expected utilities of the master against optimized attack strategies (W_1).

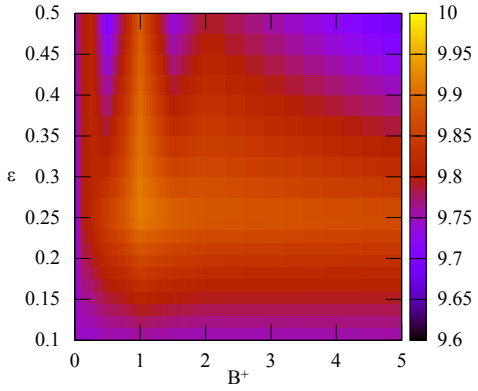


Figure 4.15: Expected utilities of the master combined for different scenarios (W_1).

few attackers, these need to be even more reliable than trustworthy workers in order to be competitive enough to be selected. However, this is only the case if attackers have full control over their error rate, as we have assumedt. Even if this is not the case in practice, attackers will apparently not increase their profit by performing worse than trustworthy workers.

Higher Fractions of Attackers For higher fractions of attackers (Fig. 4.14), attackers have a better chance of being selected. This allows them to return more incorrect results. As a consequence, the master's utility becomes smaller if it is not able to detect malicious workers. For $f_a = 0.5$, the trust model performs badly if both the tolerance factor and the increase in the Boltzmann parameter are high. In the extreme case of $f_a = 0.9$, the impact of a bad configuration becomes even more evident. A small tolerance factor seems to be advisable here. The reason for a small tolerance factor is that otherwise it is probable that many incorrect results are getting accepted, because there are many malicious sources that return incorrect results. If the Boltzmann parameter is too high, the model starts to exploit what it has learned about the sources too early: before it has found the trustworthy sources. So, when there are many malicious sources, the exploration phase should take longer in order to find the few trustworthy sources.

Table 4.6: Excerpt of restricted payoff matrix, with utilities averaged over several settings.

s_T			$\mathcal{U}'_T(s_T, \text{br}^*(s_T))$
B^+	ϵ	W	
0.0	$\frac{1}{10}$	W_1	9.73883
⋮	⋮	⋮	⋮
1.0	$\frac{1}{4}$	W_1	9.91838
⋮	⋮	⋮	⋮
5.0	$\frac{1}{2}$	W_3	9.70634

Combination of Results In practice, only one of the scenarios occurs, but at this point it is not known which one. Therefore, to make the trust model robust against all scenarios, the master's utilities from the different scenarios have to be merged. As mentioned before, we propose to use a weighted average over the utilities, where the weights[8] define how robust the model should be in which scenario. Additionally, the weights can reflect how probable each scenario is assumed to be. Lower fractions of attackers seem to be more probable in general. However, due to the possibility of a Sybil attack (see Sect.2.2.2), high ratios of attackers are still possible. Addressing these two issues, we accordingly weight the four scenarios $f_a = (0, 0.1, 0.5, 0.9)$ with the weights $(0.3, 0.3, 0.2, 0.2)$ respectively. Figure 4.15 shows the combined expected utilities; note that a different scale is used for utility to make the structure more clear. From these values, we can derive a trust model configuration with optimal expected utility. A tolerance factor of $\epsilon = 0.25$ and a B^+ between 1 and 2 seems to yield the highest utility for the master. This can be verified by looking at the restricted payoff matrix as it is illustrated in Table 4.5. An excerpt of this matrix filled with the results from the experiments is shown in Table 4.6; $\mathcal{U}'$ denotes the utility combined from the different scenarios. In this table the highest expected utility is framed; the optimal trust model configuration can be read from this row. The complete matrix can be found in Appendix D.

[8] not to be confused with the weights of the trust model W_i.

Chapter 5

Collusion Detection for Redundant Requesting

In this chapter, we propose an algorithm to detect collusion among malicious information sources. Colluding information sources can become a problem when replication is used to ensure correctness of acquired information. The proposed algorithm is a complement to replication-based schemes for information acquisition. The basic idea of the algorithm is to measure how similar the information from different information providers has been in the past, and cluster the sources accordingly; honest sources will form a tight cluster, which makes it possible to identify colluders.

Organization In Section 5.1, we discuss the problem of collusion in the context of replication. We detail our model and assumptions in Section 5.2, and propose an algorithm for collusion detection in Section 5.3. Following this, we first provide a theoretical analysis of the algorithm in Section 5.4, and then evaluate it experimentally in Section 5.5.

This chapter is related to [147, 146, 152].

5.1 Redundant Requesting and Collusion

"Redundant requesting", which is also called "replication" or "redundancy", works as follows. A requester makes the same request $\mathcal{Q}$ to several information sources redundantly; he applies majority or plurality voting to the set of returned replies $\{\mathcal{R}_i\}_i$ to decide in favor of the information that appears most often. This technique is for instance used in volunteer computing to check the correctness of returned results [10, 105], or in P2P systems to eliminate incorrect information (e.g., in [81]).

Redundant requesting is based on the assumption that the different selected sources are independent from each other; if this is the case, the probability that a majority of sources returns the same incorrect information becomes in general negligibly small. However, if the sources collude, this independence assumption is violated; this is for instance the case if an attacker controls several sources. Then, if a majority of colluding sources is selected, these can collectively return the same incorrect result, which will be accepted by the requester. To better understand the significance of this collusion threat, we have to look at the probability of randomly picking a majority of colluders for a vote. Let k be the

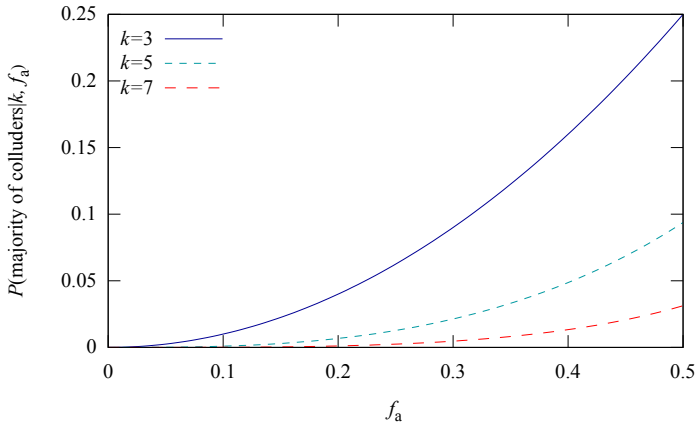


Figure 5.1: Probability of selecting a majority of colluding sources.

redundancy level, i.e., the number of sources to which one request is made redundantly. We only consider odd redundancy levels to make it less probable to get tie votes; let m_k denote the number of sources needed to form a majority in a vote. Furthermore, let f_a denote the fraction of colluding sources in the population from which the sources are picked. Then, the probability of picking a majority of colluding sources can be approximated[1] as follows:

$$P(\text{majority of colluders}|k, f_a) = \sum_{i=m_k}^{k} {f_a}^i \cdot (1 - f_a)^{k-i} \,. \tag{5.1}$$

Figure 5.1 shows the resulting probabilities for different redundancy levels. For higher redundancy levels, the probability of selecting a majority of colluders by chance decreases. However, to keep the overhead low, a low redundancy level is preferable; a common redundancy level in current systems is $k = 3$ [11]. In addition to this, against the background of the *Sybil attack* (see Sect. 2.2.2), high rates of colluding sources become more likely to occur in practice. Under these circumstances, it becomes likely that a majority of colluders will be randomly picked, as can be seen from the figure.

Against this background, countermeasures against collusion become necessary. For this reason, we propose an algorithm that detects colluding information sources in this context of replication with majority/plurality voting. The basic idea is to cluster the information sources based on how similar the information that they have returned in the past has been. The *honest* sources, i.e., those that do not collude, will form a tight cluster; in the same way, the set(s) of colluding sources will form tight clusters. Under the assumption of having less than 50% of colluding sources, the cluster of honest sources will be the largest, and all sources that do not belong to this cluster are suspects.

5.2 Model and Assumptions

In order to obtain one piece of information, the same request is made redundantly to k information sources. We call such a set of sources a *vote*. For any new vote, the sources are randomly selected from

[1] This assumes an infinitely large set of sources $\mathcal{S}$. For a finite number of sources, sampling without replacement would need to be considered. However, for our purposes the differences are negligible.

the overall set of all available sources; this set is denoted by $\mathcal{S}$. For the sake of simplicity, we assume that this set $\mathcal{S}$ is static over time, i.e., sources do not enter or leave the system. We discuss the impact of such dynamics later in Section 6.3.1. Furthermore, we allow any fraction of colluding sources in $\mathcal{S}$ smaller than 0.5. This assumption is needed later on for deciding which clusters contain the malicious and which contain the honest sources.

5.2.1 Plurality Voting

In our model of redundant requesting, plurality voting is applied in order to determine which reply is accepted as correct: A piece of information is accepted iff it has been returned by more sources than any other piece of information. Thus, a piece of information can also be accepted if it does not have an absolute majority. On this note, we define *plurality* and its counterpart *minority* as follows:

Definition 5.1 (Plurality and minority)**.** *The* plurality *of a vote is the cardinality of the strictly largest group of sources that returns identical replies. If there is no such group, then there is no plurality. All sources that do not form the plurality of the vote, and only those sources, form the* minority*.*

As stated above, only odd redundancy levels k are considered, in order to reduce the probability of ties in votes. The number of nodes needed to get an absolute majority in a vote is then given by $m_k = \left\lceil \frac{k}{2} \right\rceil$.

In practice, honest sources also may return incorrect information, even if with a very small probability, denoted by *error rate* ρ. To account for this, we model honest sources with an error rate of $\rho = 0.00221$ with reference to [88]. We therefore distinguish between correct and incorrect honest sources in a vote. For the sake of simplicity, we make the assumption that if the replies of two honest sources are both incorrect, then they are different. In our theoretical analysis, this assumption makes the detection of colluders harder because it decreases the correlation among honest sources. As a consequence, the results of the analysis would be better if the assumption were not made. In other words, honest sources with correlated error rates are not a problem for our collusion detection mechanism. To redundant requesting however such sources appear like colluders and so would be a problem. In volunteer computing systems, the correlation of errors can for instance be caused by several honest workers using the same type of faulty CPU (e.g., remember the Pentium FDIV bug [104]). However, Kondo et al. [88] showed that failures of hosts are not correlated in general.

5.2.2 Attacker models

We assume that colluding sources can efficiently communicate with each other and can rapidly reach collective decisions. This assumption is based on the Sybil attack scenario in which a single attacker creates several identities and makes them appear as distinct individuals. Therefore, colluders can decide collectively which wrong result to return. We consider the following types of colluders:

Unconditional colluders (UC) Try always to collude, i.e., each time a colluder is involved in a vote, he returns the same incorrect result as other colluders in this vote.

Conditional colluders (CC) Collude *only* if they know that at least an absolute majority of sources in the vote are colluders. Otherwise they return the correct information.

Unconditional camouflage colluders (UCm) Decide with a certain probability p_{cam} collectively for each vote whether to collude.

Conditional camouflage colluders (CCm) Decide with a certain probability p_{cam} collectively for each vote in which they form the majority, whether to collude. In all other votes they return the correct information.

Unconditional colluders will be in the minority of a vote whenever there are more correct honest sources in that vote than colluders. Conditional colluders only cheat if they know for sure that they will be in the majority. Therefore, conditional colluders are basically as effective as unconditional colluders. But at the same time, they are harder to detect, because in all votes where they most probably would not succeed in cheating, they behave as if they were honest. However, conditional collusion requires the colluders to know the redundancy level k: In general, colluders can only know how many other colluders are in the vote, but not how many honest sources are involved; hence, they know whether they would collude successfully only if they know the number of sources in the vote, i.e., the redundancy level. Thus, CC and UC are sophisticated strategies for the scenarios where attackers know k and where they do not know k respectively. Via p_{cam}, camouflage colluders can exactly define how often they want to collude; a lower probability makes it harder for the algorithm to detect them, but also makes their collusion less effective. Actually, UC and CC are two special cases of camouflage colluders, namely for $p_{\text{cam}} = 1$. We explicitly defined the two attacker models because they are the most extreme forms of collusion; however, we need to do the theoretical analysis only for the two camouflage colluders. For the sake of completeness, we should mention attacks in which attackers yield a high correlation to some selected honest sources, in order to build "bridges" between the cluster containing the honest sources and the cluster(s) of the malicious sources. These attacks are not feasible under the assumption that attackers cannot know the identities of the honest sources involved in a vote, because then they cannot build such bridges to selected honest sources. This assumption is reasonable, because the sources are not interacting with each other, and the communication from the requester to the sources can be secured.

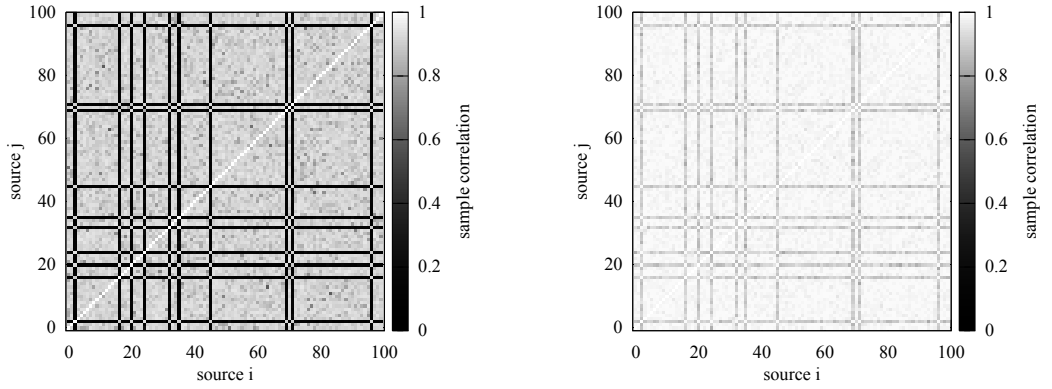
5.3 Collusion Detection Algorithm

Our aim is to identify colluders within a given set of sources $\mathcal{S}$. To this end, we use a distinctive feature, which we call "correlation", that reveals the borderline between the set(s) of colluders and the set of honest sources. This distinctive feature is based on the following observation.

For attackers, the only effective way to attack a redundant request is to collude, i.e., to collectively return the same incorrect result. Colluders will succeed in votes where they hold the plurality. But in such votes, all the honest sources will form the minority. Now, we can count for each pair of sources how often they have been together in the same group (plurality/minority), and how often they have been in opposite groups. As we show later, the counts for a pair of sources provide statistical evidence about the relation between the two sources, i.e., whether they are both malicious, both honest or one is malicious and one is honest. We use the counts to estimate the distinctive feature, namely the correlation. This feature has the desirable property that, no matter by what collusion strategy, colluders cannot influence the correlation between pairs of honest sources. Moreover, a colluder can increase his correlation with an honest source only by withdrawing from collusion.

Definition 5.2 (Correlation)**.** *The* correlation p_c *of two information sources A and B is the conditional probability that, given that A and B are together in a vote, both are in the same group (plurality/minority) of that vote. More formally this is:*

$$P(\textit{A and B are in the same group of a vote} | \textit{A and B are together in that vote}) \ . \tag{5.2}$$



(a) UC, 10 votes/pair

(b) CC, 100 votes/pair

Figure 5.2: Sample collusion matrices ($k = 3$, $|\mathcal{S}| = 100$).

From this definition it follows that for a series of n independent votes where both sources participate, the number of votes in which they are in the same group is binomially distributed with parameters n and p_c.

5.3.1 Estimating Correlation

For a pair of sources, we can count the number of votes where both participate, denoted by n, and the number of votes where both are in the same group, denoted by y. Then, following the same derivation as in Section 4.2.1.1, we find that the best correlation estimate is given by:

$$\hat{p}_c = \frac{y+1}{n+2}\,. \tag{5.3}$$

We can estimate the correlation for each pair of sources from $\mathcal{S}$. This results in what we call a *correlation matrix*, a symmetric matrix of size $|\mathcal{S}|^2$, containing the correlations between all pairs of sources. Figure 5.2 shows two example matrices for the cases UC and CC respectively. The UC-matrix has been computed after 10 votes per pair of sources on average; for the CC-matrix 100 votes have taken place. In both settings there are 10 colluders, and these have the same indices. The dark areas indicate low correlations. The rows and columns that contain mostly low correlations show the colluders, because these correlate much only with other colluders; therefore only the crossings are bright. This allows us to use clustering to draw a line between honest sources and colluders. In the UC-case, the borderline is much more obvious, which suggests that clustering will yield better results in this case.

5.3.2 Clustering

Intuitively, the correlation between two sources reflects how similarly they "vote". This can be helpful to us if it implies something about the similarity in their attitude. That is, if correlation is *high* for pairs of colluders and pairs of honest sources, and *low* for mixed pairs consisting of one colluder and one honest source. We later show in Section 5.4 that this is in fact the case. Therefore, we can interpret the correlation as a measure of similarity. This allows us to cluster the set of sources $\mathcal{S}$ based on the correlation matrix.

For our scenario, we have to deal with the following facts:

- The number of clusters cannot be fixed in advance. This is because if we fixed the number of clusters to some l, then $l-1$ attackers could try to correlate very little with any other source, so as eventually to form $l-1$ different clusters; all other attackers would then probably be in the biggest cluster, together with all the honest sources. Hence, l has to be determined anew for each clustering.
- Inputs to the algorithm are similarities; no points in a Euclidean space are given.
- The matrix we are dealing with does not contain zero entries, i.e., it is not sparse.

Many different algorithms for cluster analysis have been proposed in the literature. There are different classes of such clustering algorithms that can deal with the latter two facts. One of these classes are the "linkage methods" from the area of "hierarchical clustering" [78]; in this approach, clusters are build incrementally, and one can stop for example if the next two clusters to be merged are too far away from each other ("distance criterion"). Another type of clustering is called "spectral clustering" [164]. Here, a number of clusters l has to be specified; there are approaches to determine a good l for a given dataset (e.g., see [164]). Alternatively, if the similarity values are normalized to the common correlation interval $[-1, 1]$, one can apply "correlation clustering" [14, 35], which does not require l to be specified. Finally, the sources can be interpreted as vertices, and the correlation values as edge weights, in an undirected graph; then, a "graph clustering" algorithm can be applied (e.g., see [20]). It is out of the scope of this thesis to discuss which of the numerous clustering schemes will be the best in our case. Instead, we want to show that there is some scheme that is able to detect colluders following different collusion strategies.

5.3.3 Algorithm

Algorithm 1 Collusion Detection

1: **procedure** DETECT-COLLUSION($\mathcal{S}$,$H_{\mathcal{S}}$ [,P])
2: $M \leftarrow$ COMPUTE-CORRELATION-MATRIX($H_{\mathcal{S}}$)
3: $\{C_1, \ldots, C_k\} \leftarrow$ CLUSTER(M[, P])
4: $C_{max} \leftarrow \max(C_1, \ldots, C_k)$ ▷ Select largest cluster
5: $S \leftarrow \mathcal{S} \setminus C_{max}$ ▷ Take all but largest cluster
6: **return** S ▷ Return suspects
7: **end procedure**

The collusion detection algorithm is shown in Alg. 1. It takes as inputs: the set of sources $\mathcal{S}$ in form of their IDs, the set $H_{\mathcal{S}}$ that contains the voting history of each pair of sources, i.e., the number n of votes in which both participated, and the number y of these votes in which they were in the same group (plurality/minority), and finally an optional parameter for the clustering algorithm P; whether this parameter P is needed and its form depend on the specific clustering scheme used. With the voting histories, the algorithm can compute the correlation matrix M (line 2). Based on the correlation matrix, the clustering is performed (line 3). The largest cluster is presumed to contain the strongly correlated honest sources, so it is selected (line 4) and subtracted from $\mathcal{S}$ (line 5). The remaining set of sources S is returned as set containing the suspect sources (line 6).

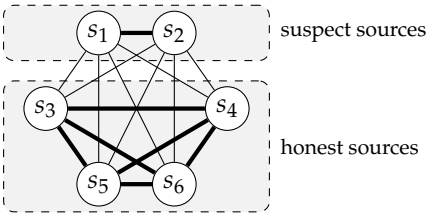


Figure 5.3: Application of the algorithm to a very small example graph ($|\mathcal{S}| = 6$).

Example 5.1 (Partitioning)**.** *Figure 5.3 illustrates for a very small example set of sources, how the algorithm should partition the sources into honest and suspect ones; the set of sources is represented in form of a complete graph where thick edges represent high correlations, and thin edges represent low correlations. As can be seen from the graph, the strongly correlated honest sources s_3, s_4, s_5 and s_6 should form a separate cluster, leaving s_1 and s_2 as suspects.*

5.4 Theoretical Analysis

Algorithm 1 interprets the correlation between two sources as their similarity. In this section, we analyze whether the correlation is actually a good similarity measure; this is the case if it measures two colluders to be similar, two honest sources to be similar, and a colluder and a honest source to be dissimilar. We first show how we can theoretically compute correlation for these three types of pairs, and then analyze its properties concerning the similarity.

5.4.1 Computing Correlation

To compute the correlation, we do not use the common *correlation coefficient* [107, p. 103], because it measures the linear relationship between quantitative variables, and we are dealing with qualitative variables (to be or not to be together in the same group). Instead, we use enumerative combinatorics to compute p_c. For the sake of simplicity, we assume a large set of sources $\mathcal{S}$, and thus can use sampling with replacement. The error introduced, in comparison with sampling without replacement, is negligible for our purposes.

In the following, we demonstrate how p_c can be computed for a pair of *two honest sources* (we write "honest&honest" or simply "h&h"). We only address the case of unconditional camouflage collusion (UCm) here. The derivations of all other formulas, i.e., those for the pairings honest&colluder (h&c) and colluder&colluder (c&c), as well as for CCm can be found in Appendix E.1.

The computation of p_c depends on certain parameters, which are listed in Table 5.1. To compute p_c in the case considered, we add up the probabilities of all mutually-exclusive cases in which the two honest sources are together in the same group. Because we account for the fact that honest sources can also return incorrect information, a number of cases have to be considered.

Given that they are in the same vote, two honest sources A and B are also together in the same group (plurality/minority), iff any of the following is the case:

- they both return the correct information: $P_1 = (1-\rho)^2$.

Table 5.1: Parameters for computing correlation p_c.

k	redundancy level
f_a	fraction of colluders in $\mathcal{S}$
p_cam	probability that camouflage colluders actually collude
ρ	probability that a honest source returns incorrect information
p_h	probability of picking a correct honest source, i.e., $p_h = (1-f_\mathrm{a})(1-\rho)$
p_w	probability of picking an incorrect honest source, i.e., $p_w = (1-f_\mathrm{a})\rho$

- they both return incorrect information: $P_2 = \rho^2$.
- exactly one of them returns the correct information: $P_3 = 2 \cdot \rho \cdot (1-\rho)$, and the correct source is not in the plurality, which means that either:
  - apart from them there are only incorrect honest sources: $P_4 = {p_w}^{k-2}$, or,
  - there are at least as many colluders as there are appearances of the correct information in the vote, and the colluders actually decide to collude. Apart from the two honest sources A and B, there are $k-2$ remaining sources in the vote. For this part of the vote, let index i count the number of colluders, and let j count the correct honest sources. Then, the remaining $k-2-i-j$ sources are incorrect honest sources. Now, only cases have to be considered in which the number of colluders is at least as high as the number of correct honest sources, i.e., where $i \geq j+1$. This is the case if $j \leq i-1$; but since there are only $k-2-i$ sources left in the vote, it must hold $j \leq \min(i-1, k-2-i)$. This gives:

$$P_5 = p_\mathrm{cam} \cdot \underbrace{\sum_{i=1}^{k-2} \binom{k-2}{i} {f_\mathrm{a}}^i}_{\substack{\text{prob. of picking} \\ i \text{ colluders}}} \cdot \underbrace{\sum_{j=0}^{\min(i-1,\, k-2-i)} \binom{k-2-i}{j} {p_h}^j {p_w}^{k-2-i-j}}_{\substack{\text{prob. of picking } j \leq i-1 \\ \text{correct honest sources}}} . \tag{5.4}$$

These cases are mutually exclusive, and so, we can compute correlation p_c by adding the corresponding probabilities:

$$p_c^{UCm}(h\&h) = P_1 + P_2 + P_3 \cdot (P_4 + P_5) . \tag{5.5}$$

5.4.2 Correlation as Similarity

To assign the sources to clusters that contain only colluders *or* honest sources, these two groups of sources have to be separated. This is illustrated in Figure 5.4. In this section, we want to analyze whether correlation is a good measure for this purpose. This is the case, if the correlation is low for h&c pairs, which would for instance be all (h_i, c_j) in the figure; and high for h&h and c&c pairs, i.e., the pairs (h_1, h_2) and (c_1, c_2) in the figure. To this end, we computed the correlations as described in the previous section. Figure 5.5 shows these correlations for h&h, h&c and c&c pairs in the basic case of redundancy $k = 3$ and an error rate $\rho = 0.00221$. For h&c pairs, the camouflage probability has been varied; for h&h

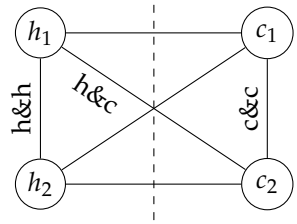
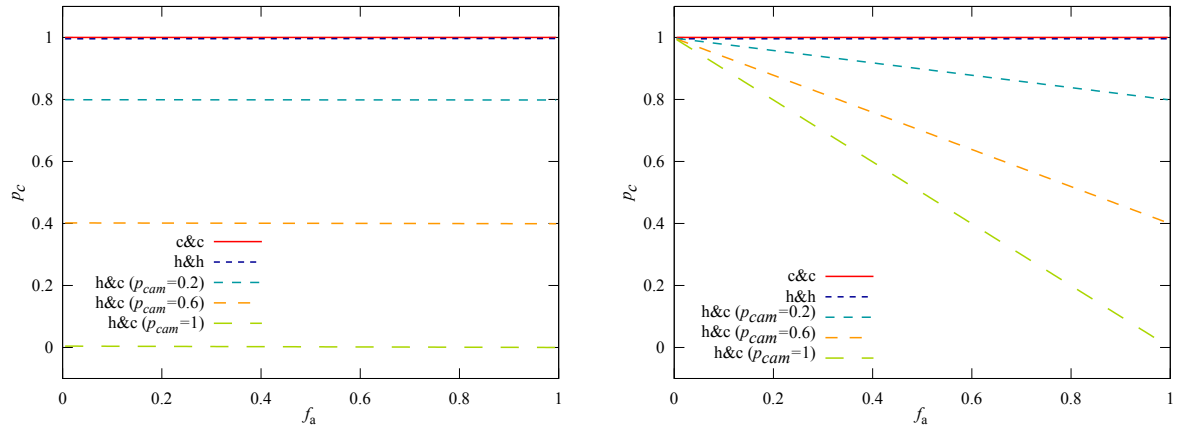


Figure 5.4: Dividing line between honest sources (h_1, h_2) and colluders (c_1, c_2).



(a) UCm (b) CCm

Figure 5.5: Correlations between different types of sources ($k = 3$, $\rho = 0.00221$).

and c&c pairs this has not been done because the differences are negligible in this case. Further cases are addressed in Appendix E.1.3. The figure reveals that in fact the correlation of h&c pairs is generally lower than the correlation of h&h and c&c pairs. This is the case both for CCm and UCm colluders. Furthermore, it holds for both types of colluders that if they collude more often, the correlation for h&c pairs is lower, and thus the distance to the correlation of other pairs becomes greater. Generally, one can say that the greater this difference, the easier it is to detect the dividing line between honest sources and colluders. Therefore, in the CCm case one can expect that the smaller fractions of colluders f_a, the less accurate will be the clustering. To increase the difference in the correlations between h&h and h&c pairs in the CC-case, it can help to exponentiate the correlation matrix. We call this "amplification", and discuss its effects in Appendix E.2. To summarize, correlation is a suitable similarity measure with the desirable property that the more harm the colluders attempt, the easier they can be detected.

5.4.3 Undetectable Attack

There is a collusion strategy, which cannot be detected by our algorithm. This strategy is to collude only if *all* sources in a vote are colluders. As with the CCm strategy, this strategy requires the attackers to know the redundancy level. This means that the attack can be countered by varying the redundancy level. Another measure against this attack is to increase the redundancy level. The success probability of the attack is ${f_a}^k$. For example, for a fraction of colluders of $f_a = 0.5$ and a redundancy level of $k = 5$,

this is only $\frac{1}{2^5} = 0.031$ and for $k = 7$ only 0.008. The success probability decreases even further for decreasing f_a.

5.5 Evaluation

We evaluated the algorithm for two different clustering algorithms, both from the field of *graph clustering*: the *Markov Cluster Algorithm* (MCL) [159] and *Minimum Cut Tree Clustering* [47], which we call "MinCTC". In [20], MCL has been shown to be more accurate than several other algorithms. The authors did not test MinCTC and so we compare it to MCL in this work.

5.5.1 Accuracy

5.5.1.1 Choice of Clustering Parameters

In the following, we describe how the clustering parameters of MCL and MinCTC were chosen for the experiments.

MCL MCL is based on the simulation of stochastic flows in graphs. The only parameter for MCL is the *inflation parameter* $I \in [2, 30]$. This parameter determines the level of granularity of the clusterings obtained. The experimental results show that the accuracy of the algorithm does not depend greatly on the exact choice of this parameter. As discussed in Appendix E.2, we amplify the correlation values $\hat{p}_c$ with an additional parameter θ. Since MCL uses a probabilistic interpretation of the edge weights, the relative differences between the correlation values are generally amplified.

MinCTC MinCTC is based on the construction of *minimum-cut trees* in graphs, as defined in [66]. The nodes in such a mincut tree correspond to the nodes in the original graph. Weights are attached to the edges in a mincut tree in such a way that the minimum edge weight on the path between each pair of nodes in the tree corresponds to the minimum cut for these two nodes in the graph. In [62], the authors experimentally compared different approaches for constructing mincut trees. In our work, we use the *GHs* implementation (a variant of the Gomory-Hu algorithm [66] using the source selection heuristic) because it turns out to be more robust than the alternatives [62].

MinCTC clusters an input graph in the following way. In th first step, the graph is extended by an auxiliary vertex, which is connected to all other vertices by new edges. For these edges, a specific edge weight is used, this being the only parameter of the algorithm. For the resulting graph, the mincut tree is constructed. Then the auxiliary node and all edges connected to it are deleted from the tree. Finally, each tree in the resulting forest is returned as a distinct cluster.

Through experimentation we found that, depending on the setting, there is a particular range of suitable parameters. Below this range, the algorithm returns only a single cluster containing all sources (underfitting); in these cases, the auxiliary node in the mincut tree is a leaf and so no mincut crosses the auxiliary node. Above this range, it returns as many clusters as sources, i.e., each source makes up its own cluster (overfitting); this is because the auxiliary node connects to every node in the mincut tree and so the mincut of every pair of nodes crosses the auxiliary node. Thus, the algorithm may find suitable clusterings only for parameters within the range – if it is assumed that there are colluders among the sources. Therefore, we implemented a *binary search* to find a parameter that lies in this range and produces a specified number of clusters k (we used $k = 2$). The search is terminated and the output of the algorithm is accepted if it returns more than one cluster and fewer clusters than there

are sources. Sometimes the range is empty, because either there are not enough observations, or the rate of collusion is too low. If the search is not successful, all sources are classified as honest in order to avoid false positives. As with MCL, we amplify the correlation values in the experiments with different exponents θ (again see App. E.2).

5.5.1.2 Accuracy Measure

To measure the accuracy of the algorithm, we first label the output of the algorithm according to the confusion matrix described in Section 2.5.2; this results in a number of *true positives* (TP), *true negatives* (TN), *false positives* (FP) and *false negatives* (FN). The positives are identified suspects, and negatives are identified honest sources. As accuracy measure, we use the F_1-Measure [126], which is the harmonic mean of *precision (P)* and *recall (R)*; these are defined as follows:

$$P = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} \,, \tag{5.6}$$

$$R = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \,. \tag{5.7}$$

Thus, the precision represents the fraction of correctly identified colluders among all sources marked as suspect, and recall represents the fraction of correctly identified colluders among all sources that are in fact colluders. In other words, precision is the probability that a suspect is in fact a colluder, and recall measures the completeness, i.e., how many of the colluders have been detected. In the best possible case, both are 1, that is $FP = FN = 0$. We check the accuracy using the F_1 measure, which is defined as follows:

$$F_1 = \frac{2 \cdot P \cdot R}{P + R} = \frac{2 \cdot \mathrm{TP}}{2 \cdot \mathrm{TP} + \mathrm{FP} + \mathrm{FN}} \,. \tag{5.8}$$

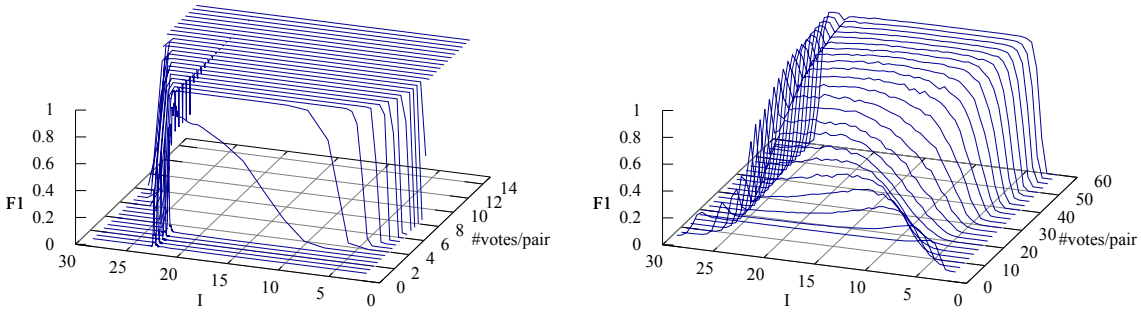
F_1 takes a value of 1 iff our algorithm returns a set of suspects that contains all colluders but no honest sources. If F_1 takes a small value then FP and/or FN are high and TP and/or TN are small. We will later also use the "false positive rate" (FPR) to examine false positives separately. This rate describes the fraction of honest sources that are falsely marked as suspects among all honest sources; it is thus computed as follows:

$$FPR = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TP}} \,. \tag{5.9}$$

5.5.1.3 Results

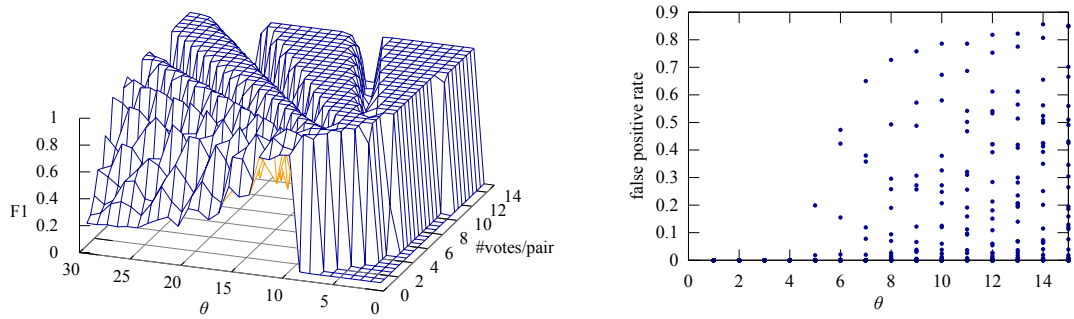
The algorithm is evaluated for the setting described in Section 5.2 for $|\mathcal{S}| = 100$ and $f_\mathrm{a} = 0.1$. Whether the algorithm will perform better or worse in other scenarios follows from the theoretical analysis. We set the redundancy level k to a common value of 3 [11]. Each run is averaged over 10^3 randomly-generated sets of voting histories $H_\mathcal{S}$. Each history $H_\mathcal{S}$ is based on a certain number of observed votes. This number is indicated for the simulation results as an average number of observed votes per pair of sources, denoted by "#votes/pair". The overall number of observed votes can then be computed as follows. For x votes/pair and sources $\mathcal{S}$, we have to multiply x with the number of edges in a complete graph that has the nodes in $\mathcal{S}$ as edges; so the overall number of observed votes is: $x \cdot \frac{|\mathcal{S}| \cdot (|\mathcal{S}|-1)}{2}$.

Figure 5.6(a) shows the accuracy of the algorithm using MCL for the UC case. No amplification θ has been applied. An average of $F_1 = 1$ first occurs for an average number of 4 votes/pair and inflation



(a) UC (b) CC, $\theta = 20$

Figure 5.6: Accuracy of collusion detection algorithm (MCL, $f_a = 0.1$).



(a) F_1 (b) False positive rate

Figure 5.7: Accuracy of collusion detection algorithm (MinCTC, UC, $f_a = 0.1$).

parameter I between approximately 13 and 22.8. For more than 10 votes/pair, the algorithm provides perfect accuracy for all possible parameters. MCL showed problems for a small range of parameters $22.8 < I < 23.3$. For this set of parameters, MCL did not always converge within 5 seconds, and if it converged returned a particularly high ratio of false positives, which points to overfitting by MCL. Figure 5.6(b) shows the accuracy of the algorithm using MCL for the CC scenario. An almost perfect precision of $F_1 = 1$ on average was not achieved until around 60 votes/pair, a very high value. MCL showed similar problems to the UC case for $22.5 < I < 23.3$. For smaller amplification, the accuracy became worse, and for $\theta = 1$ the algorithm was not able to detect colluders. This confirms the findings of Appendix E.2 that amplification helps in the UCcase.

Figure 5.7(a) shows the accuracy of the MinCTC variant of Alg. 1 for varying amplifications θ. Generally speaking, the algorithm needs around 4 votes/pair to reach perfect accuracy for a specific θ. Starting from $\theta = 5$, several "valleys" can be observed, in which the accuracy drops dramatically. This is possibly caused by the procedure that we used to find an optimal clustering parameter; consequently, other procedures might help to prevent these valleys. With few votes/pair, the algorithm struggles, especially for higher amplifications. Figure 5.7(b) plots the false positive rate against the amplification for all experiments. This shows that a high amplification makes the rate worse in the UC-case; this corresponds to our findings in Appendix E.2. But also, the figure reveals that for an amplification of $\theta < 5$, the algorithm never returned false positives, which is a nice property. The accuracy of the algorithm using MinCTC for the CC scenario is not shown, because for up to 200 votes/pair and varying θ it throughout

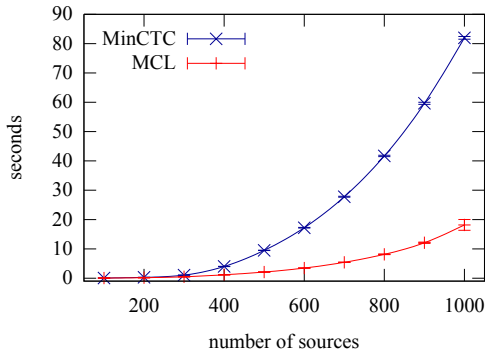


Figure 5.8: Running times of MCL and MinCTC (UC, 10 votes/pair, $\theta = 4$).

performed with an average of $F_1 = 0$; that means that it never identified any colluder as suspect. Apparently, MinCTC cannot cope with cases where h&c pairs differ only slightly from other types of pairs. This may also be due to the problem of finding a suitable parameter for MinCTC.

5.5.2 Computational Complexity and Running Time

Theorem 5.1. *The complexity of Algorithm 1 is in $\mathcal{O}(|\mathcal{S}|^2)$ iff the complexity of the clustering algorithm used is also in this complexity class; otherwise it is in the same complexity class as the clustering algorithm.*

Proof. Line 2 of Algorithm 1 computes the correlation for each pair of sources, and thus is in $\mathcal{O}(|\mathcal{S}|^2)$. The complexity of line 3 is determined by the complexity of the graph clustering algorithm used. Lines 4 to 6 of the algorithm are in $\mathcal{O}(|\mathcal{S}|)$. □

MCL's complexity is essentially dominated by matrix multiplication [20] and is in $\mathcal{O}(|\mathcal{S}|^3)$, but can be reduced by implementation to worst case $\mathcal{O}(|\mathcal{S}|c^2)$ [159], where c is a pruning constant smaller than $|\mathcal{S}|$. MinCTC relies on maximum flow algorithms which are also polynomial in complexity [47]. Thus, the overall complexity is at least in $\mathcal{O}(|\mathcal{S}|^2)$, but depending on the clustering algorithm it can be higher; if for example MCL is used it is in $\mathcal{O}(|\mathcal{S}|c^2)$.

We experimentally compared the running times of MCL and MinCTC for different sizes of $\mathcal{S}$. In the tests, both algorithms take their inputs from a file but do not write any output. The input mainly consists of the sample correlation for each pair of sources, and so the file size grows nearly quadratically with the number of vertices in a graph. However, the two algorithms worked on files with the same size and so differences in their runtime are attributable entirely to the processing performed. Figure 5.8 shows the running times of MCL and MinCTC in seconds for the same setting (UC, 10 obs./edge); experiments were run on a 2.33 GHz Linux machine with an Intel Xeon CPU and 3GB RAM. Although MCL was shown in [20] to be relatively slow, it still clearly outperforms MinCTC and its complexity seems to grow considerably more slowly.

5.6 Dealing with Suspects

In this chapter, we proposed an algorithm that detects colluding behavior when using redundant requesting. The detection of colluders is only a first step. As soon as suspected sources have been detected, measures have to be taken to:

- recheck the results that the suspected sources have returned in the past, and
- prevent the suspected sources from submitting further incorrect results.

As proposed by Silaghi et al. [138], the rechecking can be done by additional redundant requests to unsuspected sources. However, this again makes the scheme sensitive to collusion, because it carries the risk of consulting malicious sources that have been honest previously. Therefore, an explicit verification of the results on trusted machines is more advisable. To prevent the suspects from cheating in the future, they can be blacklisted. A drawback of this approach is that, if the classification was incorrect, honest sources are removed from the pool of potential sources, and so their capacity is lost. Alternatively, in addition to redundant requesting, one can spot-check these sources frequently. A third approach is to include them in votes with sources that belong to the strong cluster of honest sources. Again, this carries the risk that some of the sources in the strong cluster are malicious sources that have pretended to be honest. Finally, one could treat suspects more carefully by attaching a weight to their vote, and setting this weight considerably lower than that of honest sources. It is outside the scope of this thesis and subject to further studies which of these approaches is in practice the most suitable.

Chapter 6

Related Work and Comparisons

In this chapter, we first point to research that is closely related to our work. We then compare the approaches that we have proposed in this thesis with each other, and discuss common issues.

Organization In Section 6.1, we discuss related work. We provide a comparative study of our mechanisms in Section 6.2, and discuss common issues in Section 6.3. Finally, we describe in Section 6.4 a concrete example from the literature that illustrates how spot-checking and redundant requesting can be combined.

6.1 Related Work

Even though we have taken an abstract perspective in this thesis, our work is closely related to work that has been done on sabotage tolerance in volunteer computing and other forms of distributed computing such as *grid computing* [48]. Therefore, we first review significant related literature from these fields of research. We then compare a mechanism for collusion detection from the literature with our algorithm. Finally, we take a look at work that has been done in the field of trust.

6.1.1 Spot-Checking

Sarmenta [133] proposes *combining* redundancy with conventional spot-checking. In this approach to spot-checking, workers are regularly assigned work units for which the correct result is already known. Workers that get caught returning incorrect results are blacklisted and their past results are discarded (if identities can be checked). In their model, attackers are described by a Bernoulli process and return correct and incorrect results with a specific probability. Assuming this attacker model, the author attains probabilistically guaranteed levels of correctness. However, Sarmenta's model does not consider attackers that follow more sophisticated strategies such as those that we have considered in our work, e.g., collusion or repetitive cheating. More precisely, the model assumes the workers' error rates to be constant over time, which is not the case for instance in volunteer computing systems [88].

Zhao et al. [185] propose the "Quiz" scheme for result verification in peer-to-peer grids. In this scheme, a worker delegates a whole package of work units to a single worker. In such a package, quizzes for which the correct result is known are interspersed. This is similar to our spot-checking approach with challenges. However, Quiz discards a whole package if only one of the quizzes is answered incorrectly, reduces the trustworthiness of the respective worker and possibly blacklists it. This makes it unsuitable

for scenarios where also honest workers exhibit small error-rates greater than zero. In our work, we have shown that our trust model reacts at least as reasonably and sometimes better to different types of sources than their trust models. Additionally, Zhao et al. find that under collusion, Quiz outperforms redundancy; in the absence of collusion however, Quiz performs worse. This argues for the usefulness of our trust model, or – in cases where collusion cannot be precluded – redundancy with collusion detection.

Germain-Renaud and Monnier-Ragaigne [60] propose a result-checking algorithm for volunteer computing systems. They use Wald's sequential test to incrementally check results from a large batch of jobs until either the batch can be accepted or has to be rejected. As with our method, this technique is applicable in cases where a certain error rate in the acquired information is acceptable; the authors apply their technique for instance to check a large-scale Monte-Carlo simulation. However, the technique requires what they call an "oracle", which recomputes the results at runtime. On the one hand, this keeps the number of checks small, because the acceptance decision can be made early. On the other, in scenarios apart from volunteer computing, such as file-sharing networks, an oracle that verifies a piece of information at runtime is not generally available. Therefore, we have proposed a mechanism that uses prepared challenges, which can for instance be known facts, to check newly-acquired information. As a consequence, and in contrast to the oracle-based mechanism, our mechanism requires that the number of challenges is determined in advance, and that challenges are indistinguishable from real requests. Roch and Varrette [127] propose a spot-checking mechanism for *dependent* tasks in the scenario of divide-and-conquer computations. Independently of the attacker model, they reach a user-defined error probability bound for detecting massive attacks, where the bound depends on the minimum ratio of malicious workers in the population. Again, correct results for the spot checks are assumed to be recomputed reliably.

Du et al. [41] propose an effective spot-checking mechanism for scenarios where many computations have to be performed but only few of the results have to be returned. This is for instance the case when using distributed computing to break a password in a brute-force fashion. In their approach, a worker constructs a "Merkle-tree" that combines the hashes of all results belonging to a specific task in a tree-like manner. The worker commits the root of the tree to the master. The master can now spot-check the worker and reconstruct the root of the tree with the hashes, together with their siblings that also have to be transferred. This approach can be used to make sure that a worker computes all intermediate steps in a chain of dependent computations, where only the final result has to be submitted. However, it cannot be used to ensure the correctness of a set of independent pieces of information, which is the actual research question of this thesis.

6.1.2 Collusion Detection

Silaghi et al. [138] propose an algorithm for detecting *collusion*. They consider three different types of malicious workers: first, naive workers that randomly return incorrect results with a certain probability; secondly, workers that only return incorrect results if they know that they hold the majority in a vote; and third, a mixture of the first two types. For each worker, their core mechanism counts the number of votes that were cast against it in past votes. Then, since naive workers have a different probability of votes being cast against them, this type and some part of the third type can be identified and removed from the votes. In order to spot the second type of colluding worker, they again apply the redundancy principle to test workers in the remaining "conflicting" votes. At this point, we face again the problem of collusion. This is especially relevant in the case of high fractions of colluders, where the probability of getting false negatives is high. In contrast, our collusion detection mechanism does entirely without

making use of redundancy checks. Also, as we have shown, correlation can directly identify conditional collusion, something not accomplished by the measure of Silaghi et al. However this comes at a cost. In an optimal case, our algorithm needs around 4 votes/pair to reach an acceptable classification error (see Sect. 5.5.1.3); this means overall $4 \cdot \frac{100 \cdot 99}{2} = 19,800$ votes. In a comparable scenario, the algorithm of Silaghi et al. works already with only around 3,000 votes. The reason for this is that they look at the behavior of single sources (and there are $|\mathcal{S}|$ sources), while we look at the correlation between sources, that is at pairs of sources (and there are $\frac{|\mathcal{S}| \cdot (|\mathcal{S}|-1)}{2}$ pairs). To summarize, our algorithm addresses more attack strategies than the algorithm of Silaghi et al., but it needs also a much higher number of observed votes. The improvement of this required number of observed votes is a topic for further work.

Another, very severe, drawback of Silaghi et al.'s approach is that they assume that the colluders are not aware of being observed by their mechanism. Otherwise, malicious sources could use strategies that make them look like honest sources. Our collusion-detection algorithm works without this assumption: A colluder cannot impact the correlation between two honest sources, and so cannot dissolve the strong cluster of honest sources; the only strategy that brings a colluder closer to this cluster of honest sources is to withdraw from collusion.

6.1.3 Trust Modeling

Two main directions in computational trust modeling have been pursued in the literature. On the one hand, there is the cognitive approach, promoted mainly by Castelfranchi and Falcone [29, 30, 46]. In their approach the trustor tries to develop a "theory of the mind" of the trustee in order to reason about how trustworthily he will behave in future interactions. On the other hand, there are the computational models. Here, the trustor uses evidence in the form of experiences to derive numerical values that should reflect the trustworthiness of the trustee. In the majority of cases, this process involves probabilistic mechanisms [167, 26, 28, 45, 157] or formulas that are based mainly on intuition [110, 174, 182, 130, 74]. Literature surveys on further trust models can be found in [12, 117, 131, 85].

Our trust model, which belongs to the second category, computes the trust and uncertainty values based on the mean and the variance of the Beta distribution. Many other evidence-based trust models also use the beta distribution to this end [109, 167, 157, 87, 172, 23, 79, 111]. However, the way in which the experiences are processed, and how trust and uncertainty values are used for decision-making, differs from model to model.

6.1.3.1 Trust in Information Sources

Fullam et al. [55] propose a method for information acquisition from possibly malicious or incompetent sources. They show how to manage the trade-off between costs for information acquisition, quality of the acquired information and the coverage of an agent's goals. In their model, the reliability of an information provider is assessed by checking whether some acquired information fits the agent's beliefs. In contrast to our mechanisms, their approach is not able to handle acquired information that is not related to an agent's beliefs, as is the case for results of mathematical calculations, music or video files, etc. Further papers deal with the assessment of the trustworthiness of sources based on input from human users [61, 70]. One requirement for our mechanisms was however that they work without the need of human intervention.

Liau [97] formalizes the relationships between *belief*, *trust* and *information acquisition* by defining an extended modal logic. Trust itself is modeled by a modal operator $T_{ij}\varphi$, which means that agent i believes in agent j's judgment on proposition φ. A proposed axiomatic system makes it possible to

derive complex properties of trust in the context of information acquisition. In [34], Liau's logic is extended to capture the concepts of *topics* and *questions*. The extension with topics allows the context-sensitivity of trust to be taken into account; the extension with questions makes it possible to formalize queries in the form $(\mathcal{Q}, \mathcal{R})$, as introduced in the introduction.

Demolombe [36] formalizes different definitions of trust by means of modal logics. To this end, the author differentiates essentially four properties of an information source – a source is called:

- "sincere" towards some agent A, iff it believes everything that it tells A to be true.
- "credible", if everything it believes is a fact.
- "cooperative" towards some agent A, iff it tells A all its beliefs.
- "vigilant", if it knows all existing facts.

With reference to Section 4.1.3, our definition of trust accounts explicitly for sincerity and credibility; it implicitly addresses the notions of cooperativeness and vigilance, because we require that the provided information $\mathcal{R}$ makes $(\mathcal{Q}, \mathcal{R})$ a fact, i.e., $\mathcal{R}$ contains all correct answers to $\mathcal{Q}$ (see Sect. 1.1). Besides this, since we are dealing with probability theory, our definition accounts not only for complete trust or distrust, but also for estimates in between. In a paper which proposes an algorithm that performs belief revision while taking into account the trustworthiness of the sources, Barber and Kim [16] give another definition for trust in information sources. They define trust as "the confidence in the ability and intention of an information source to deliver correct information". This also corresponds to our definition. However, it ignores the subjective nature of trust. Also, it does not include the property of cooperativeness as defined by Demolombe.

6.1.3.2 Combining Evidence

For cases where the most recent evidence is not compatible with older evidence, Xiong and Liu [177] propose considering only the latest evidence, and dropping the old evidence. In their approach, a trust value t' is computed on a subset of all available evidence that contains only the most recent evidence; if t' is considerably *smaller* than the trust value that is based on all the evidence, t' is used as a final trust value. This means that a trustee can easily lose its trustworthiness through a few bad actions, but rebuilding the trust is hard. Like ours, their idea aims to counter the strategic behavior of a malicious trustee. However, in our case, we are dealing with a separate set of challenges N and real requests M, and thus, it can happen that by chance all incorrect replies fall on N, exceeding the area defined by threshold ϵ, making the most recent error rate estimate fall too low. So, if x is low, $\hat{\rho}$ can become high, even for honest sources. For this reason, we take also into account older error rate estimates that are based on conflicting evidence, since they might actually have been more accurate. For the same reason, we also start a new set of evidence if the new error estimate is *smaller* than before. This makes it possible for an information source to increase its trust value very fast with a single set of correct replies – something Xiong and Liu wished to prevent. However, in our case, the uncertainty will increase as well, and thus the source does not actually benefit from such a behavior. The following example illustrates the difference between their approach and ours.

Example 6.1. *Assume threshold* $\epsilon = \frac{1}{12}$ *for combining evidence, and weights* $(w_i)_i = (1, 3, 6)$. *An honest information provider has error rate* $\rho = \frac{1}{10}$. *Five requests with* $m, n = 10$ *are made to this provider. Assume the incorrect replies* y_i *to the challenges are in chronological order* $(1, 2, 0, 1, 5)$. *In our model, for the first four replies, the evidence would be aggregated, because the corresponding error rate estimates* $\frac{2}{12}$, $\frac{3}{12}$, $\frac{1}{12}$ *and* $\frac{2}{12}$ *are*

within $[\frac{2}{12}-\epsilon, \frac{2}{12}+\epsilon] = [\frac{1}{12}, \frac{3}{12}]$. *However, the last one would fall outside the range* ($\frac{5+1}{10+2} = \frac{1}{2} > \frac{3}{12}$) *and so the overall trust value would be computed as follows (note that* w_1 *is not used since there are only two error rate estimates available):*

$$\dot{t} = \frac{w_2 \cdot \hat{\rho}[1,4] + w_3 \cdot \hat{\rho}[5,5]}{w_2 + w_3} = \left(3 \cdot \frac{5}{42} + 6 \cdot \frac{6}{12}\right) / 9 = \frac{47}{126} \,. \tag{6.1}$$

In Xiong and Liu's approach, the trust value for the whole evidence is first computed:

$$\left(1 \cdot \frac{1}{12} + 3 \cdot \frac{2}{12} + 6 \cdot \frac{6}{12}\right) / 10 = \frac{43}{120} \,. \tag{6.2}$$

The trust value based on the last reply only is $\frac{1}{2}$, *and because it is greater than* $\frac{43}{120} + \epsilon = \frac{53}{120}$, *only this value, which we designate* t', *is used. For reasons of comparison, let us reduce it to the same denominator as the trust value from our model, which gives* $t' = \frac{1}{2} = \frac{63}{126}$. *The estimate of our model is more accurate in this case:*

$$|\dot{t} - \rho| = |\frac{47}{126} - \frac{1}{10}| < |\frac{63}{126} - \frac{1}{10}| = |t' - \rho| \,. \tag{6.3}$$

Again, one could argue that our trust value would be less accurate if the source was actually malicious. This is true for the trust value in isolation; however, our trust value comes along with an uncertainty value, and this uncertainty value reflects the sudden change: It increases from 0.157 *after the fourth reply, to* 0.253 *after the fifth reply.*

6.1.3.3 Trust Model Tuning and Evaluation

Many evidence-based trust models from the literature can be parameterized. We will now describe the parameters used by a number of models. Buchegger and Le Boudec [23] use several aging factors, which correspond to the weights used in our model, and give a rule of thumb for how to choose them. Similarly, Wang and Vassileva use "learning rates" [166]. Capra and Musolesi [28] use aging factor-like parameters that have to be defined subjectively. In the FIRE model [74], a "recency scaling factor" (an aging factor) and the "temperature" (the Boltzmann parameter in our model) are used. Other models [172, 85, 87, 176] also use tunable aging factors, and [75] uses an *aging function*. In [186], a modifiable threshold determines when an individual can be considered as trustworthy. Parameters for balancing trust and reputation values are used in [161, 160, 13]. The issue of how to systematically tune all these parameters has, to our knowledge, not yet been addressed.

The *ART testbed* [54] was developed to enable a comparison of the various trust- and reputation models proposed in the literature. In this testbed, several agents, each of which employs a certain trust model, compete in a turn-based game. The performance of an agent is measured both from an agent- and a system perspective according to certain objective criteria. Thus, the model allows measurement of the performance of different trust models competing with each other. An important limitation of this approach is that it is hard to generalize the performance measurements: Would the trust model perform similarly if it competed with another selection of trust models? The performance of a trust model has always to be seen in the context of the other trust models involved. Therefore, the testbed is not suitable as a general benchmark. In contrast to this, our tuning procedure aims for expected utilities for different scenarios of cheaters. This makes it suitable as the basis for a benchmark, in which the expected utilities of different trust models can be compared.

6.2 Comparative Study

In this section, we compare trust-enhanced spot-checking with redundant requesting plus collusion detection in terms of applicability, accuracy, robustness and overhead.

6.2.1 Applicability

As discussed in Section 3.1.4, spot-checking relies on the creation of challenges. Clearly, if this is not possible, (trust-enhanced) spot-checking cannot be applied. Furthermore, our spot-checking mechanism requires that a source can in general deal with $m + n$ requests at once. In contrast, redundant requesting requires only one single request to be processed by one source. However, redundancy requires that a set of k sources can handle the same request, which is not a requirement of spot-checking. Also, the application of the collusion detection algorithm requires that a high number of requests is made to a preferably small number of sources. Although such a situation is also favorable to the trust model because the characteristics of the few sources can be learned very well, it is not a strict requirement.

6.2.2 Accuracy

Our collusion detection algorithm is not equipped with functionality to deal with suspects (see also Sect. 5.6). For this reason, we cannot compare its accuracy with that of the trust-enhanced spot-checking mechanism. However, two findings from the literature point to the fact that redundant requesting is only more effective than spot-checking if collusion can be prevented. Yurkewych et al. found that, if collusion cannot be prevented, redundancy is much more costly than non-redundant requesting [181]. Confirming this, Zhao et al. [185] found that their spot-checking mechanism outperforms redundant requesting under collusion; in the absence of collusion however, their mechanism performs worse. This argues for further research in both fields, that is, trust-enhanced spot-checking and collusion detection.

6.2.3 Robustness

The robustness of the two approaches depends strongly on the fraction of malicious sources in $\mathcal{S}$. The highest tolerable fraction for the collusion detection mechanism is $f_a < 0.5$. In principle, the trust model can tolerate higher fractions; however, exploration has to be emphasized in such cases.

The two approaches are quite different, and so are the attack strategies. The trust model has been shown to deal with all attack strategies considered in this thesis. However, for unknown attack strategies, the model has to be evaluated anew. The most effective attack against redundant requesting is collusion. To defend against this attack, we have proposed the collusion detection algorithm. Although not every kind of colluding behavior can be detected by this algorithm with few observations, the more the attackers collude, the better the algorithm detects them.

If challenges can be produced reliably, the trust model can handle incorrect information also in cases where incorrect information is correlated across honest sources. Also, the collusion detection algorithm is able in this situation to distinguish honest sources from colluders. However, as discussed in Section 2.3.2, redundant requesting would struggle here. More precisely, it would accept correlated incorrect information that is provided by a majority of honest sources. Therefore, in scenarios where errors are commonly correlated over honest sources, the trust model is preferable to redundant requesting; alternatively, additional mechanisms that detect the correlation of errors have to be put in place.

Finally, there is an attack that could be performed by *saboteurs* (see Sect. 4.1.3.1). When such an attacker controls a large fraction of sources, he can implement a "denial of service" (DOS) attack, both

against the trust model and redundant requesting. To this end, malicious sources return random results all the time. For the the trust model, this would extend the time that it takes to find the trustworthy sources. For redundant requesting, there would be many more conflicting votes. However, in both cases, the error rate of accepted results would not suffer. This is because replies with an unacceptable estimated error rate in the case of the trust model, and conflicting votes in the case of redundant requesting, are ignored.

6.2.4 Overhead and Complexity

Spot-checking has an overhead of at most $\frac{m+n}{m}$ (see Sect. 3.3.2.2). For the trust model this cannot be easily measured, because it depends on the competence and intention of available sources, but also on how "trustful" the model is set to be (eq. (4.17)). In any case, it is bound by $\frac{m+n'}{m}$, where n' is the maximal number of challenges. The overhead of redundancy is k. That means that the trust model can start with $n' = m \cdot (k-1)$ challenges to have at most the overhead of redundancy; to give an example, if redundancy is set to a common value of $k = 3$, the trust model can start with 20 challenges for 10 real requests. Considering the subsequent decrease in challenges needed for trustworthy sources, even more challenges can be used initially.

Let us now analyze a hybrid approach that combines replication with spot-checking. Here, replication can be used to fill a pool of challenges. As soon as this pool contains n challenges, the spot-checking mechanism is used. In other words, replication is used to generate the challenges. In this manner, we can make a definite gain compared to the use of replication alone as we will now show; although note that the information providers have to be selected carefully to avoid the possibility of intersection attacks (see Sect. 3.3.1.2). The use of redundancy gives n results for $k \cdot n$ requests. These results can be used as challenges to get m more results through spot-checking. We thus get overall $m + n$ results for $k \cdot n$ requests, which gives an overhead of:

$$\text{Overhead}(\text{hybrid}) = \frac{k \cdot n}{n + m} \,. \tag{6.4}$$

For example, for $n/m = 1$ and $k = 3$, we halve the overhead from 3 for pure redundancy to $\frac{3 \cdot n}{m+n} = \frac{3}{2}$.

Concerning complexity, all three mechanisms, i.e., the spot-checking mechanism, the trust model and the redundant requesting, are linear in the number of requests. However, the collusion detection mechanism is at least quadratic, and its current implementation cubic, in the number of sources. Depending on when and how often the collusion detection mechanism is executed, the complexity of replication with collusion detection can be considered to be much higher.

6.3 Common Issues

In this section we discuss two issues that are related both to trust-enhanced spot-checking and redundant requesting.

6.3.1 High Churn Rates

In our trust model tuning procedure, we used a fixed population of sources. The rate with which new sources enter the population and old sources leave it, known as the "churn rate", can impact the effectiveness of the Boltzmann exploration. If the churn rate is high, a more intense exploration is advisable

for two reasons: More new sources have to be encountered, *and* the already-known sources are likely to leave the system. Therefore, to tune a trust model for a concrete P2P network, knowledge about churn rates in the network should be integrated in the tuning procedure. This can be done by using churn models [154] to simulate typical churn, and tuning the model under these circumstances.

Similarly, the churn rate would have an impact on the performance of our collusion detection mechanism. If sources leave the system very often and reenter each time with a new identity, our algorithm becomes unsuitable because it requires a high number of observations on sources with constant identity. Under the assumption that sources are interested in keeping the requester as a "customer", they do however have an incentive to keep their identity, so that only malicious sources would be interested in changing their identity (whitewashing attack, see also Sect. 4.1.2.4). As a consequence, these would move even further apart from the strong cluster of honest sources. Unfortunately, newcomers that are actually honest also would be initially far away from the large cluster of honest sources, and would need some time before eventually becoming members of the cluster of honest sources.

6.3.2 Dealing with Inhomogeneous Sources

Throughout this thesis we have made the assumption that information is either correct or incorrect. However, in practice there are sometimes slight variations in what is correct. In the scenario of volunteer computing, Taufer et al. [156] have for example been facing the problem that the CPUs of different workers produced slightly different results. To counter this problem they have proposed outsourcing computational tasks only to "compatible" CPUs. The mechanisms proposed in this thesis assume that all considered information sources are compatible in that sense. For example, we assumed for redundant requesting that, when picking k workers, these are selected from homogeneous subsets of workers. As a consequence of this consideration, the collusion detection mechanism has to be applied for each subset of homogeneous sources separately. Alternatively, and this is applicable to spot-checking as well, an acceptance range can be defined within which a piece of information is considered to be correct. This idea can be applied to computational results, and also for instance to music files. If different encoding algorithms are used to convert audio data from a CD to MP3 files, the resulting files will probably not be identical but still might be acceptable for a user. In this case too, based on some similarity measure for audio data, acceptable deviations can be defined.

In [173], an attack on a specific CAPTCHA called "reCAPTCHA" was reported. This attack exploits the fact that reCAPTCHA considers a reply to a challenge to be correct if it is in a certain Levenshtein distance [94] from the correct reply. This shows that the definition of an acceptance range for correct information can enable new attacks, and thus should be avoided if possible.

6.4 Combination

An illustration of how the ideas of spot-checking and redundant requesting can be combined is given by the reCAPTCHA system [163]. ReCAPTCHA uses distributed human effort to transcribe old documents. The old documents are scanned and largely recognized by optical character recognition (OCR) algorithms. The system uses different OCR algorithms and accepts a word if it is recognized as the same word by all of them. The words on which the OCR algorithms disagree are passed to a pool of unrecognized words. In order to use human input to recognize these words, the system employs CAPTCHAs (see also Sect. 2.2.2). Such a CAPTCHA presents two words to the user, one unrecognized word and one for which the correct transcription is known. This is basically a special case of our spot-checking

algorithm from Chapter 3 with one real request and one challenge – they call a "control word" what we call a "challenge". If the user answers correctly the challenge, the reply to the real request is recorded as plausible. If a plurality of votes, and a minimum of 2 humans and one OCR, returned the same word to a real request, the word is finally accepted. Since the mechanism uses plurality voting, it is principally susceptible to the collusion of human users. A group of colluding humans would have to maintain a collective database containing requests and the corresponding incorrect replies. To make this attack effective though, the group would have to be quite large. Although this is very unlikely to happen in the case of parsing old documents, because there is simply no obvious incentive to do so, it might become relevant for other applications of this principle. Apart from that, we think it could be interesting to use a trust model in order to ignore the input of unreliable humans here; as in our proposed trust model, the replies to challenges could be used as evidence in this case. An identification of a human by for instance his IP address would be a prerequisite for that.

Chapter 7

Conclusions

7.1 Summary

In this thesis, we have dealt with the problem of malicious information sources in distributed information systems. We considered the case where the acquired information cannot be verified directly. The research objective was to come up with mechanisms that identify trustworthy sources. The scope was set to objective information with a binary degree of truth. Below, we first summarize the three main chapters. Following this, we restate our main contributions by answering the research questions that we have asked in the introduction. Finally, we give some concluding remarks.

7.1.1 Spot-Checking with Challenges (Chapter 3)

In the first part of this chapter, we have proposed a mechanism that allows estimation of the error rate of acquired information without explicitly checking it. This is done by mixing the real requests with unidentifiable challenges, for which the correct replies are known beforehand. The more challenges are used, the more accurate is the estimate. Therefore, an optimal number of challenges emerges from a trade-off between the costs accruing from preparing the challenges, and taking the risk of having an inaccurate estimate. We argued that a uniform prior should be used in the estimates, because it equally addresses different kinds of probabilistic attack strategies. Intersection attacks can be rendered impossible by always using the same challenges for the same real requests, and using new challenges for new real requests. The complexity of a request is in $\mathcal{O}(n)$.

In the second part of the chapter, we showed how challenges can be used to assess the accuracy of alternative classification components in an intrusion detection system at runtime. This allows for selecting the most accurate components but also for directing the intrusion detection system so that it pays more attention to the most harmful network attacks. To this end, threats have to be modeled in form of attack trees, and the expected damage of a threat has to be defined. Based on this, challenges can be selected in a way that they better cover the most harmful attack trees. Experiments suggest that this helps to improve the performance of the system for the selected threats.

7.1.2 Modeling Trust in Information Sources (Chapter 4)

In this chapter, we first have developed a definition for "trust in information sources", and have analyzed certain difficulties when computationally modeling trust. We have then proposed an evidence-

based trust model that helps both to decide which source to select next, and to reduce the overhead of spot-checking. In this model, trust is represented in form of a trust value and an uncertainty value, for which we proved the compliance with the two properties demanded by Wang and Singh [167]. We evaluated our trust model against different types of information sources: By comparing the model to other models from the literature, we found that it reacts appropriately to the different source types. Additionally, we have shown that a reduction of challenges is more cost-efficient if the creation of a challenge is relatively cheap. Finally, we have extended the trust model with Boltzmann exploration, and tested it in dynamic settings. We found that the performance of the trust model (i.e., how fast it converges to select trustworthy sources exclusively), depends mainly on how the trust model is configured. This stresses the importance of the configuration on the performance of a trust model. For this reason, we have proposed a generic procedure that optimizes the parameters of evidence-based trust models. The procedure uses a genetic algorithm to search for an attacker's best attack strategy to any given trust model configuration. By looking at the expected utilities of all configurations against the best attack strategies, one can eventually select the configuration with the highest expected utility for the trust model user. An interesting additional finding from our experiments is that there are different optimal attack strategies for different fractions of attackers: If there are only few attackers, they have to be much more cautious, because it is harder for them to compete with the high number of trustworthy sources; they can be more aggressive if they are numerous.

7.1.3 Collusion Detection for Redundant Requesting (Chapter 5)

In this chapter, we have introduced an algorithm for collusion detection that can be applied to redundant requesting. A theoretical analysis has revealed in which scenarios the algorithm is able to detect colluders. In general, both unconditional and conditional colluders can be identified by means of correlation, the former more easily than the latter. We showed how amplification can help to improve the algorithm's chances of success for conditional colluders. To provide a proof of concept, we evaluated a concrete implementation of the algorithm using two different graph clustering algorithms, MCL and MinCTC. We found that the number of observed votes is crucial to the accuracy of the algorithm: For unconditional colluders the number can be relatively small; for conditional colluders many observations are required. MCL clearly outperforms MinCTC both in terms of running time and accuracy. The algorithm's complexity is determined by the complexity of the clustering algorithm used, but is at least quadratic.

7.1.4 Summary of Main Contributions

In the following, we restate the research questions asked in the introduction and answer them concisely. We provide references to the respectively most relevant section.

Q1: *How may we apply spot-checking if the creation of spot checks at runtime is infeasible?*

By merging real request with challenges, verifying the replies to the challenges, and based on this, estimating the error rate to the real requests. (See also Sect. 3.1)

Q2: *What is the optimal number of challenges for a given number of real requests?*

The number of challenges that minimizes the expected costs both for the creation of the challenges needed and for the expected error made in the estimate. (See also Sect. 3.2)

Q3: *What does it mean to "trust in an information source?"*

To believe that the source both is able to make sure that the information it provides is correct, and that it does so. (See also Sect. 4.1.3)

Q4: *What information can be used as evidence for a source's trustworthiness?*

All failures that are not excusable, and all successes that are not strategic. (See also Sect. 4.1.2.3)

Q5: *How may we build trust in an information source if we cannot directly verify the information acquired from it?*

By spot-checking a source with challenges and using the resulting error rate estimates as statistical evidence for (un-)trustworthiness. (See also Sect. 4.2)

Q6: *How may we systematically choose optimal parameters of evidence-based trust models?*

By selecting the configuration that yields the highest expected payoff against all of its best responses. (See also Sect. 4.3.2)

Q7: *How may we detect collusion attacks against redundancy with plurality voting?*

By measuring how similar sources voted in the past, clustering them accordingly, and marking all sources as suspects that do not belong to the largest cluster. (See also Sect. 5.3.3)

7.2 Strengths and Challenges

Although spot-checking allows for an exact adjustment of the overhead, redundancy still prevails in volunteer computing. The reason for this might be that in this scenario sufficient sources are available to mean that the overhead is not the decisive argument. However, there is evidence that spot-checking is less costly if collusion is possible. In this thesis, we therefore explored both directions for sabotage tolerance: the improvement of trust-enhanced spot-checking, and the extension of redundant requesting with the capability of detecting collusion. While the former direction aims to reduce the resulting *overhead*, the latter aims to optimize the *robustness*. More precisely, the trustworthiness of a source can be used to reduce the number of spot checks; however, our evidence-based trust model is generally sensitive to first-time cheaters, and in the implausible worst case it can be deceived by chance. In contrast, redundant requesting is very robust, provided that collusion attacks are prevented and errors are not correlated across honest sources; on the other hand, the overhead of redundant requesting must be at least 100%. Below we discuss further benefits and limitations of the two approaches. A combination of the two approaches seems to be very promising, and is discussed in the section on further work.

We have tested the robustness of our trust model against various attack strategies. We found that on average it handles them appropriately. Still, this is no proof of the presumption that the trust model is robust against other attack strategies as well. This is a pointer to a general need for common methodologies of evaluation in the field of computational trust modeling. Although many evidence-based trust models have been proposed in the literature, no generally accepted benchmark for trust models has been suggested. As a first step, we have developed a procedure that determines an optimal configuration for a trust model. The proposed tuning procedure assumes an attacker's utility function to be known and that attackers are perfectly rational. Under these assumptions, we can measure the expected utility of a trust model against *all* attack strategies considered in the search space. Furthermore, this allows trust models to be compared in terms of utility, and so can be seen as an initial benchmark. However, because

genetic algorithms are used in the procedure, and these do not guarantee that a global optimum will be found, special attention has to be paid to the configuration of the genetic algorithm.

During the application of the tuning procedure we found that if there are more attackers, they will use more aggressive attack strategies to maximize their utility. This is because it takes longer for the trust model to explore the set of potential trustees. This shows that the Sybil attack can also be a problem for a trust model that individually assesses the trustworthiness of different sources: Just by creating a higher number of malicious sources, an attacker has more chances of going undetected in trials while submitting incorrect information. Thus, measures against the Sybil attack are necessary here too.

Having an effective collusion detection algorithm would make attacks other than denial-of-service attacks harmless. Our collusion detection algorithm is able to detect collusion attacks that would otherwise have a high chance of success. Nevertheless, it suffers from two drawbacks. First, it requires a high number of observed votes before it can make accurate classifications; this becomes especially unfavorable when there are high churn rates (see Sect. 6.3.1). Secondly, due to the fact that collusion is only detected after it has happened, the affected results have to be discarded or rechecked. Therefore it is crucial to strive for a reduction of the number of observed votes required.

7.3 Further Work

In the following, we identify avenues for possible continuation of research relating to the three main chapters of this thesis.

7.3.1 Spot-Checking

To extend the spot-checking mechanism, degrees of truth could be considered, e.g., information being almost correct. This would for instance be the case if the challenges are taken from previously accepted replies, which may only be considered to be correct with a certain probability. The error rate estimation would have to be adapted (see also Sect. 3.1.4). Furthermore, the assumption of an objective truth could be relaxed to account for opinions. This would certainly move the problem closer to the problem addressed by *collaborative filtering* techniques [124]. Finally, knowledge about a source could be used to adapt the assumed prior distribution over time. This could be an alternative approach to our trust model which it would be worthwhile to investigate.

7.3.2 Trust Model

Below, we describe different ways of how the trust model and its tuning procedure can be extended and improved.

Incompetence vs. Maliciousness If trustworthy sources have a low error rate ρ, as for instance in the case of volunteer computing [88], it is statistically justified to take information found to be incorrect as evidence of untrustworthiness. Beyond this, our trust model is not able to decide whether a trustee is *incompetent* or *malicious*. This is not the aim of the current trust model, because either of the two properties are undesirable for a source. Still, it might be interesting to have a closer look at this distinction, because an incompetent source might become competent over time, while a source found to be malicious once should probably never be considered again. Under the assumption that an incompetent source can be described by some error rate ρ, a statistical hypothesis test can be performed to this end. For a given sequence of encountered error rates s, the "null-hypothesis" can be formulated as follows:

H_0 **:** The sequence of error rates s has with high probability been created by a source that can be described by some error rate ρ.

If this hypothesis has to be rejected, the source is probably malicious. Methods from the field of *pattern recognition* could be helpful to see whether a source is following a particular strategy. Once such a strategy is identified, the trust value and uncertainty could be set to a very low value, or the source could be blacklisted to avoid that it is still selected during the exploration phase.

Pretenders A general and fundamental limit of evidence-based trust modeling is that no matter how much positive evidence one has gathered, one can never be perfectly certain about the trustworthiness of an entity – it might be that the entity is just pretending to be trustworthy (see Sect. 4.1.2.3 for an in-depth study of this issue). The problem is reflected by our uncertainty measure $\dot{u}$, which can never become zero. From a pragmatic point of view, a malicious source that always pretends to be trustworthy is equivalent to a trustworthy source. However, a malicious source most probably has an interest in minimizing the time for which it pretends. Therefore, it will try to find the perfect point at which to cheat; this attack strategy, which we call the "repetitive cheater", has been considered in our trust model evaluations.

Completeness of a Reply In our spot-checking based mechanisms, we assumed that sources always return all replies to a set of requests. However, in practice, a source might be able to answer only parts of a request. Time-outs can be applied to decide when a source is considered as not having returned (parts of) a reply. Since the source does not know what the challenges are, a partial reply is like a random selection of requests to which no reply is returned. Our spot-checking mechanism is also applicable in this situation, although the number of usable challenges might shrink. In order to prefer sources that give complete replies, we can introduce another criterion for the selection of a source, that measures how complete a reply is expected to be. A weight can then determine how important this criterion is in comparison to other criteria like the trustworthiness of a source. A source that does not give a reply that it knows to be incorrect should be preferred over a source that returns such a reply. For this reason, the trustworthiness should be weighted much more heavily than the completeness rating.

Alternatively, for the trust model, a request that is not replied to could be treated like an incorrect reply; however, to estimate the error rate of the replies, the missing replies should not be taken into account, because they would distort the estimate. Unfortunately, this approach would motivate sources to return replies even if they know that they are incorrect, which is not a desirable situation; this is because a missing reply to a real request would be taken as incorrect instead of unknown.

Trust Model Evaluation The evidence-based trust model proposed in this thesis is based on probability theory. At the outset, we assume that an information provider can be described by a single error rate ρ; in practice, for instance in the volunteer computing scenario, workers can be characterized by such probabilities [88]. However, malicious sources such as a repetitive cheater cannot be described by such an error rate. For these sources, the probabilistic derivation of our trust model is no longer valid. Therefore, an experimental evaluation of the trust model especially for malicious sources was necessary. For a finite set of attack strategies, we thus compared the model to existing models from the literature. We found that it reacts better to malicious behavior, or at least as well. However, this evaluation is restricted to the attack strategies under consideration. We did not prove that the trust model performs well for all possible attack strategies. Due to the dynamics of the selection process it would be hard to theoretically

show that the model is robust against Byzantine adversaries. Therefore, we evaluated the model experimentally. We deliberately selected attack strategies, and believe that we considered the most promising ones. For any new attack strategy that is proposed though, the trust model has to be evaluated anew.

We measured the accuracy of the trust model in terms of the *average error* in the error rate estimate, together with the respective standard deviations. Alternatively, the accuracy of a trust model could be measured through the *worst-case error*. This is especially relevant when dealing with saboteurs (see Sect. 4.1.3.1). Their intention is to maximize the error in the trustor's error rate estimate. In such a case, the trust model evaluation should be concerned with the maximal error in the error estimate, rather than the average. Also, the tuning procedure has then to be adapted such as to minimize the maximal error.

Trust Model Tuning In our current approach to trust model tuning, we have used the genetic algorithm to search for a common attack strategy that is used by all attackers. It would be too complex to allow attackers to use different strategies and then to search for a constellation of attack strategies that is optimal for the attackers. Still, it would be feasible to divide the attackers into collectives that compete with each other for the optimal attack strategy. Similarly, different payoffs functions for the attackers could be considered. The tuning procedure is based on game theory to reason about the most likely attack strategies. This assumes that the utility function of the attacker is known, and that the attacker acts in a perfectly rational manner. Consideration of the cases of partially known utility functions and attackers with bounded rationality is a topic for future work.

A way to improve the trust model would be to equip it with the capability of adapting its configuration at runtime. In other words, given an initial starting configuration, how can an operating trust model use information about the environment to constantly improve its configuration. For example, fluctuations in the churn-rates of P2P systems can be used to adapt the Boltzmann parameter. Also, an increase in the error rates encountered can be an indicator of a high number of malicious sources, which argues for a lower tolerance interval.

Context-Sensitivity First of all, depending on the scenario, the trust model should be made context-sensitive, for instance to account for changes over time of what is correct, or for the areas of expertise of the sources. For instance, techniques such as *Latent Semantic Indexing (LSI)* [43], *PLSI* [72] or *Concept Indexing* [82] could be employed to allow for defining the areas of expertise on the basis of acquired natural-language text.

Reputation Systems Reputation systems should be considered for extending the trust model in scenarios where there is more than one requester. These can help, especially in the initial phase where a requester has not yet enough direct experience, to make an informed choice of future sources. The impact of using reputation information on the Boltzmann exploration approach would also be an interesting field of study.

7.3.3 Redundant Requesting and Collusion Detection

For redundant requesting, the idea of a flexible redundancy level would be an interesting approach. This would make conditional collusion harder. Furthermore, in a *combination* with trust modeling it could help to reduce the overhead. For instance, the votes of sources could be weighted by their trustworthiness. Also, the more trustworthy sources can be selected, the lower can the redundancy level be set. Attackers might want to exploit lower redundancy levels for collusion, and so adapted techniques

to detect and prevent collusion might be necessary. To further improve the accuracy of the collusion detection algorithm, alternative clustering methods or techniques from the field of *pattern recognition* could be investigated. Concerning the integration of our algorithm into volunteer computing architectures, it would be interesting to explore parallelization in order to run it in a distributed fashion with the help of trustworthy workers. Finally, as discussed in Section 5.6, procedures have to be defined for how to proceed with suspicious sources.

Appendix A

Proofs

A.1 Posterior Distribution

The following theorem concerns eq. (3.2).

Theorem A.1. *Let $X = Z - Y$. If the prior $P(Z)$ is chosen to be a binomial distribution with parameter p, then X is also binomially distributed with the same parameter.*

Proof. For arbitrary $m, n \in \mathbb{N}$.

$$
\begin{aligned}
p(z|y) &= p(y|z)p(z)/p(y) && \text{(as by eq. (3.2))} && \text{(A.1)}\\
&= \frac{\binom{m}{z-y}\binom{n}{y}}{\binom{m+n}{z}} p(z)/p(y) && \text{(as by eq. (3.4))} && \text{(A.2)}\\
&= \frac{\binom{m}{z-y}\binom{n}{y}}{\binom{m+n}{z}} \binom{m+n}{z} p^z(1-p)^{m+n-z}/p(y) && (P(Z) \text{ is binom. prior}) && \text{(A.3)}\\
&= \binom{m}{z-y}\binom{n}{y} p^z(1-p)^{m+n-z}/p(y) && \text{(canceling out } \tbinom{m+n}{z}) && \text{(A.4)}\\
&= \binom{m}{x}\binom{n}{y} p^{x+y}(1-p)^{m+n-x-y}/p(y) && \text{(replacing } z \text{ by } x+y) && \text{(A.5)}\\
&= \left(\binom{m}{x} p^x(1-p)^{m-x}\right) \underbrace{\frac{\binom{n}{y}p^y(1-p)^{n-y}}{p(y)}}_{\text{independent of } x} && \text{(rearranging)} && \text{(A.6)}
\end{aligned}
$$

At this point consider a fixed y, and let us look at $p(x|y)$, which is simply $p(z-y|y)$. Since $z = y + x$, we have $p(z|y) = 0$ for all z with $z > y + m$ or $z < y$; $p(x|y)$ concerns exactly the part in between. Then, we can see that $p(x|y)$ is proportional to $\binom{m}{x}p^x(1-p)^{m-x}$, which is the binomial distribution with parameter p. This is because the remaining part is independent of X; in fact, it only determines the position of the above mentioned interval $[y, y+m]$. Therefore, $p(x|y)$ follows the binomial distribution with parameter p, independent of what n and y is. □

A.2 Expected Damage

In the following, we show that we do not increase or decrease the overall damage when distributing the damage from attack trees to single attacks (Sect. 3.4.3).

Theorem A.2. *The definition of $D(\mathtt{a}_i)$ (eq.* (3.39)*) satisfies the following property:*

$$\sum_i D(\mathtt{a}_i) = \sum_j D(T_j) \; . \tag{A.7}$$

Proof. We first show that $\sum_i I(\mathtt{a}_i, T) = 1$:

$$\sum_i I(\mathtt{a}_i, T) = \sum_i \left(\frac{1}{|F(T)|} \sum_{\substack{C_k \in F(T), \\ \text{with } \mathtt{a}_i \in C_k}} \frac{1}{|C_k|} \right) \qquad \text{(definition of } I(\mathtt{a}_i, T)) \tag{A.8}$$

$$= \frac{1}{|F(T)|} \sum_i \sum_{\substack{C_k \in F(T), \\ \text{with } \mathtt{a}_i \in C_k}} \frac{1}{|C_k|} \qquad \text{(rearranging)} \tag{A.9}$$

$$= \frac{1}{|F(T)|} \sum_{C_k \in F(T)} \overbrace{\sum_{\mathtt{a}_i \in C_k} \frac{1}{|C_k|}}^{=1} \qquad \text{(carefully switching sums)} \tag{A.10}$$

$$= \frac{1}{|F(T)|} \cdot |F(T)| \qquad \text{(\#clauses= } |F(T)|) \tag{A.11}$$

$$= 1 \; . \tag{A.12}$$

Now, we can prove the theorem:

$$\sum_i D(\mathtt{a}_i) = \sum_i \left(\sum_j D(T_j) \cdot I(\mathtt{a}_i, T_j) \right) \qquad \text{(definition of } D(\mathtt{a}_i)) \tag{A.13}$$

$$= \sum_j D(T_j) \overbrace{\sum_i I(\mathtt{a}_i, T_j)}^{=1} \qquad \text{(rearranging/shown above)} \tag{A.14}$$

$$= \sum_j D(T_j) \; . \tag{A.15}$$

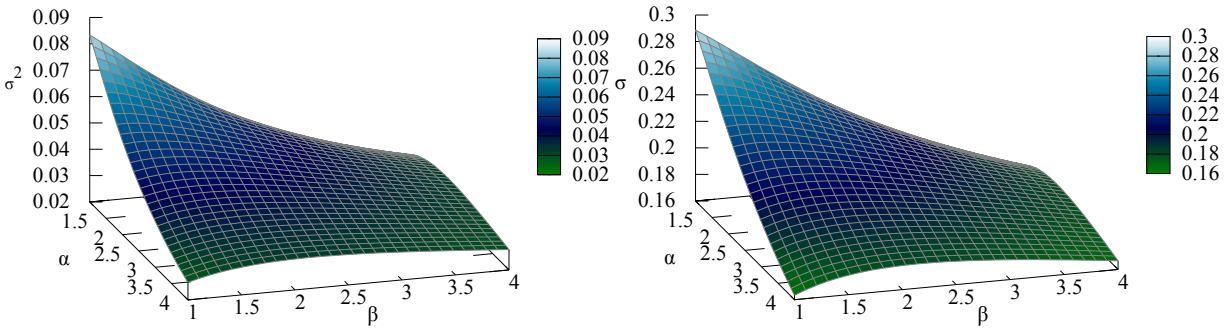
□

A.3 Uncertainty

In the following, we show that the variance of the beta distribution fulfills the two properties that – following Wang and Singh [167] – a measure of uncertainty should have.

Let α and β be the two shape parameters of the beta distribution. The variance of the beta distribution is then given by (see also Fig. A.1(a)):

$$\sigma^2 = \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)} \; . \tag{A.16}$$



(a) variance (b) standard deviation

Figure A.1: Deviations from the mean of the beta distribution.

The first theorem states that for a fixed amount of evidence and an increasing conflict within the evidence (peaking at $\alpha = \beta$), the variance strictly increases. The second theorem states that for an increasing amount of evidence and fixed conflict, the variance strictly decreases. Since the square root preserves (strict) monotonicity[1], the theorems retain their validity also for the standard deviation of the beta distribution (see Figure A.1(b)).

Theorem A.3. *For $\alpha, \beta \geq 1$ and fixed $\alpha + \beta$, the variance of the beta distribution strictly increases for increasing α up to $\alpha = \beta$ and strictly decreases afterwards.*

Proof. Let $\alpha, \beta \geq 1$ and $\gamma = \alpha + \beta$ be constant. We can express β by $\gamma - \alpha$ and get for the variance:

$$\sigma^2 = \frac{\alpha(\gamma - \alpha)}{(\alpha + (\gamma - \alpha))^2(\alpha + (\gamma - \alpha) + 1)} \tag{A.17}$$

$$= \frac{\gamma\alpha - \alpha^2}{\gamma^2(\gamma + 1)} \,. \tag{A.18}$$

The first derivative with respect to α can directly be seen:

$$\frac{d\sigma^2}{d\alpha} = \frac{\gamma - 2\alpha}{\gamma^3 + \gamma^2} \,. \tag{A.19}$$

At this point, we can see that the only extremal point is at $\alpha = \gamma/2 = (\alpha + \beta)/2$, so where $\alpha = \beta$. To prove the theorem, we only have to check that the second derivative at this point is negative and so it is a maximum at this point. The second derivative is:

$$\frac{d^2\sigma^2}{d\alpha^2} = \frac{-2}{\gamma^3 + \gamma^2} \,. \tag{A.20}$$

Clearly, the second derivative is negative for all possible $\gamma \geq 1$. □

Theorem A.4. *For $\alpha, \beta \geq 1$, and fixed conflict α/β, the variance of the beta distribution strictly decreases for increasing $\alpha + \beta$.*

[1] $\forall a, b \in \mathbb{R}^+ : a > b \Leftrightarrow \sqrt{a} > \sqrt{b}$

Proof. Let $\alpha, \beta \geq 1$ and $\xi = \alpha/\beta$ be constant. We can express β by α/ξ and get for the variance:

$$\sigma^2 = \frac{\alpha^2/\xi}{(\alpha + \frac{\alpha}{\xi})^2(\alpha + \frac{\alpha}{\xi} + 1)} \,. \tag{A.21}$$

It suffices to show that the first derivative of this variance with respect to α is negative:

$$\frac{d\sigma^2}{d\alpha} = \frac{d}{d\alpha}\left(\frac{\alpha^2}{\xi(\alpha^2 + 2\frac{\alpha^2}{\xi} + \frac{\alpha^2}{\xi^2})(\alpha + \frac{\alpha}{\xi} + 1)}\right) \tag{A.22}$$

$$= \frac{d}{d\alpha}\left(\frac{1}{\frac{\xi}{\alpha^2}(\alpha^2 + 2\frac{\alpha^2}{\xi} + \frac{\alpha^2}{\xi^2})(\alpha + \frac{\alpha}{\xi} + 1)}\right) \tag{A.23}$$

$$= \frac{d}{d\alpha}\left(\frac{1}{\xi(1 + \frac{2}{\xi} + \frac{1}{\xi^2})} \cdot ((1 + \tfrac{1}{\xi})\alpha + 1)^{-1}\right) \tag{A.24}$$

$$= \underbrace{\overbrace{\frac{1}{\xi(1 + \frac{2}{\xi} + \frac{1}{\xi^2})}^{>0} \cdot -1 \cdot \overbrace{(1 + \tfrac{1}{\xi})}^{>0}\overbrace{((1 + \tfrac{1}{\xi})\alpha + 1)^{-2}}^{>0}}_{<0} \,. \tag{A.25}$$

□

Normalization of Standard Deviation From Theorem A.4 it follows that the standard deviation σ takes its maximum value for $\alpha, \beta \geq 1$ at $\alpha = \beta = 1$, which is $\frac{1}{2\sqrt{3}}$. To normalize σ to the interval $(0, 1]$, it has thus to be multiplied by $2\sqrt{3}$.

Appendix B

Computing Margins of Error

In the following, we show how to find the margin of error for means and standard deviations. The computations are applicable in the case where both the mean (μ) and the standard deviation (σ) of the underlying distribution are unknown. Based on this, we introduce one-sided margins of error, which are used in Section 3.4.2.

B.1 Margin of Error for Means

First of all, we need to look at the following standardized variable:

$$T = \frac{\bar{\mu} - \mu}{s/\sqrt{n}} \,. \tag{B.1}$$

This variable T expresses the distance of the estimated mean ($\bar{\mu}$) from the real mean (μ) in standard deviation units. Assuming that the "underlying" probability distribution is Normal, it can be shown that this quantity T follows the *t-distribution* [141, p. 159], which only depends on n. So, we do not know μ, but we can compute the distribution of T. Now, let $t_{n-1,\mathcal{C}}$, or short t, denote the "critical value";

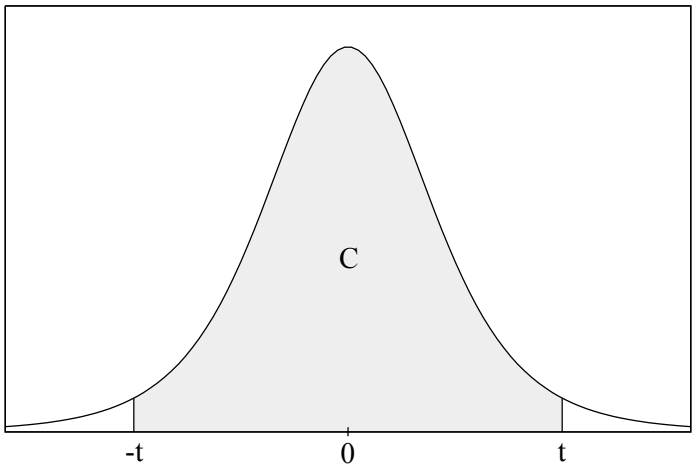


Figure B.1: Critical values for a t-distribution.

the two values t and $-\mathrm{t}$ delimit symmetrically on the right and on the left an area of size $\mathcal{C}$ under the t-distribution with parameter $n-1$ (see figure B.1). Because the t-distribution has mean 0, a random value is then with probability $\mathcal{C}$ in $[-\mathrm{t}, \mathrm{t}]$. To state that the mean μ is with confidence $\mathcal{C}$ within the interval $[\bar{\mu} - \Delta_\mu, \bar{\mu} + \Delta_\mu]$, we have to define the margin of error Δ_μ as follows:

$$\Delta_\mu = \mathrm{t}_{n-1,\mathcal{C}} \cdot \frac{s}{\sqrt{n}} \,. \tag{B.2}$$

In other words, we simply make sure that we get the right unit (compare to eq. (B.1)). The critical values can be computed, by solving $\mathcal{C} = \int_{-\mathrm{t}}^{\mathrm{t}} \mathcal{T}_{n-1}$ for t, where $\mathcal{T}_{n-1}$ denotes the t-distribution with $n-1$ degrees of freedom (see [141, p. 115]). These values can be precomputed and stored in a database. Table B.1 shows some critical values for selected n and $\mathcal{C}$. Because of the *central limit theorem*[1], the above

Table B.1: Some critical values $\mathrm{t}_{n-1,\mathcal{C}}$ of the t-distribution (taken from [107]).

$n-1$	$\mathcal{C} = 0.9$	$\mathcal{C} = 0.95$	$\mathcal{C} = 0.99$	$\mathcal{C} = 0.999$
1	6.314	12.71	63.66	636.6
2	2.920	4.303	9.925	31.60
3	2.353	3.182	5.841	12.92
4	2.132	2.776	4.604	8.610
5	2.015	2.571	4.032	6.869
		⋮		

margin of error is approximately correct for all kinds of distributions, when n becomes large enough. For smaller n, the distribution of $\bar{\mu}$ is not necessarily Normal, and so Δ_μ can get less and less accurate. However, it is the "best" we can do if the type of the underlying probability distribution and its standard deviation are not known.[2]

B.2 Margin of Error for Standard Deviations

For the standard deviation we look at the following variable:

$$S = \frac{(n-1)s^2}{\sigma^2} \,. \tag{B.3}$$

For an underlying normal distribution with variance σ^2, S is *chi-square* distributed [37, p. 278]. And so, again, although we do not know σ, we can estimate the error of s^2. And again, we need to find the two critical values that delimit an area of $\mathcal{C}$ under the chi-distribution. To cut off the same area on the left and on the right, the critical values have to cut off an area of $\frac{1-\mathcal{C}}{2}$ each, because then the area in-between is $1 - \frac{1-\mathcal{C}}{2} = \mathcal{C}$. Because the chi-square distribution is not axially symmetric, we get two different critical

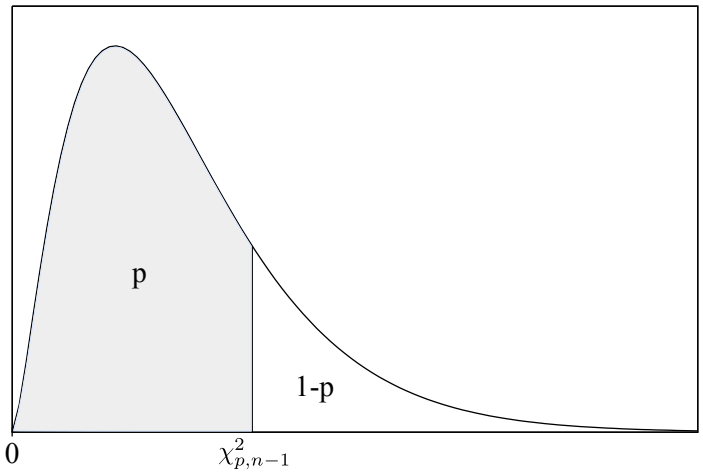


Figure B.2: Critical value for a chi-square distribution with $n-1$ degrees of freedom.

values, and later, two different margins of error. Let $\chi^2_{p,n-1}$ denote the critical value that delimits an area of p under the chi-square distribution *to the left* of the critical value (see also figure B.2). A critical

Table B.2: Some critical values $\chi^2_{p,n-1}$ of the chi-square distribution (taken from [141]).

$n-1$	$p=0.005$	$p=0.01$	$p=0.99$	$p=0.995$
1	0.0000	0.0002	6.63	7.88
2	0.0100	0.0201	9.21	10.6
3	0.0717	0.115	11.3	12.8
4	0.207	0.297	13.3	14.9
5	0.412	0.554	15.1	16.7
		⋮		

value $\chi^2_{p,n-1}$ can be computed by solving $p = \int_0^{\chi^2_{p,n-1}} \chi^2(n-1)$ for $\chi^2_{p,n-1}$, where $\chi^2(n-1)$ marks the chi-square distribution with $n-1$ degrees of freedom (see [141, p. 115]). Table B.2 shows some critical values.

The desired critical value for the left side is then simply $\chi^2_{\frac{1-\mathcal{C}}{2},n-1}$; the desired critical value for the right side is $\chi^2_{\frac{1+\mathcal{C}}{2},n-1}$, because the area to the right of it is $1-\frac{1+\mathcal{C}}{2}$ which is the desired $\frac{1-\mathcal{C}}{2}$. So, for a chi-square distributed variable S it holds:

$$P(\chi^2_{\frac{1-\mathcal{C}}{2},n-1} < S < \chi^2_{\frac{1+\mathcal{C}}{2},n-1}) = \mathcal{C}\ . \tag{B.4}$$

[1] According to the central limit theorem, the sample mean $\bar{\mu}$ is normally distributed with mean μ and standard deviation $\sigma/\sqrt{n}$ for large n [107, p. 281].

[2] Note that if we knew the actual standard deviation σ of the underlying distribution, we could use the more precise critical values z^* (see also [107, p. 355])

Substituting S (see eq. (B.3)), and solving for σ^2 gives:

$$P\left(\frac{(n-1)s^2}{\chi^2_{\frac{1+\mathcal{C}}{2},n-1}} < \sigma^2 < \frac{(n-1)s^2}{\chi^2_{\frac{1-\mathcal{C}}{2},n-1}}\right) = \mathcal{C}\ . \tag{B.5}$$

The two margins of error Δ'_σ (the left one) and Δ''_σ (the right one) can directly be seen:

$$\Delta'_\sigma = \sqrt{\frac{(n-1)s^2}{\chi^2_{\frac{1+\mathcal{C}}{2},n-1}}}\ , \qquad \Delta''_\sigma = \sqrt{\frac{(n-1)s^2}{\chi^2_{\frac{1-\mathcal{C}}{2},n-1}}}\ . \tag{B.6}$$

So, the actual standard deviation σ is with a confidence of $\mathcal{C}$ in $[s-\Delta'_\sigma, s+\Delta''_\sigma]$. As for Δ_μ, the higher n, or more Normal the underlying distribution is, the more accurate is Δ_μ.

Computational Complexity Given a sample standard deviation s, assuming that the critical values have been precomputed and are stored in tables, the computations of the margins of error, both for the mean (eq. (B.2)) and the standard deviation (eq. (B.6)), involve only single operations, and so are in $\mathcal{O}(1)$.

B.3 One-Sided Margins Of Error

In Section 3.4.2, we always make use of what we call a "one-sided margin of error". A one-sided margin of error provides the confidence that a mean (or standard deviation) is either not greater *or* not smaller than a single critical value. One can solve this by adjusting the integral boundaries for the computations of the critical values. However, we can also compute the actual confidence by adding the probability corresponding to the area, which we cut off on one side. Let $\mathcal{C}'$ be the confidence as used in the previous sections. Then, the area, which we cut off on one side is $\frac{1-\mathcal{C}'}{2}$. So, the actual confidence $\mathcal{C}$ we get for a one-sided margin of error is:

$$\mathcal{C} = \mathcal{C}' + \frac{1-\mathcal{C}'}{2} = \frac{\mathcal{C}'+1}{2}\ . \tag{B.7}$$

The other way around, if we want to have confidence $\mathcal{C}$, we need to look up the critical values for confidence $\mathcal{C}' = 2\mathcal{C}-1$. The least possible confidence we can choose is $\mathcal{C} = 0.5$ so that $\mathcal{C}' = 0$, which means that we guarantee with confidence 0.5 a value greater (or smaller) than the mean.

Appendix C

Damage of Attack Classes

Below, we show how the priorities are computed for a set of two very simple example attack trees T_1 and T_2 shown in Figure C.1 and C.2 respectively (next page). For the sake of the example, we estimate the damages of the trees to be $D(T_1) = 900$ and $D(T_2) = 50$. The two trees can be transformed into propositional logic formulae:

$$(\mathtt{a}_1) \wedge ((\mathtt{a}_2 \wedge \mathtt{a}_3) \vee (\mathtt{a}_4 \wedge \mathtt{a}_5)) \,, \tag{C.1}$$

$$(\mathtt{a}_6 \vee \mathtt{a}_7 \vee \mathtt{a}_8) \,. \tag{C.2}$$

Bringing them into canonical DNF and applying the Quine-McCluskey algorithm gives (in set notation):

$$F(T_1) = \{\{\mathtt{a}_1, \mathtt{a}_2, \mathtt{a}_3\}, \{\mathtt{a}_1, \mathtt{a}_4, \mathtt{a}_5\}\} \,, \tag{C.3}$$

$$F(T_2) = \{\{\mathtt{a}_6\}, \{\mathtt{a}_7\}, \{\mathtt{a}_8\}\} \,. \tag{C.4}$$

We can now compute $I(\mathtt{a}_i, T_j)$ for all attacks. First, let us look at $I(\mathtt{a}_1, T_1)$:

$$I(\mathtt{a}_1, T_1) = \frac{1}{|F(T_1)|} \left(\frac{1}{|C_1|} + \frac{1}{|C_2|} \right) = \tfrac{1}{2} \left(\tfrac{1}{3} + \tfrac{1}{3} \right) = \tfrac{1}{3} \,. \tag{C.5}$$

Analogously we obtain:

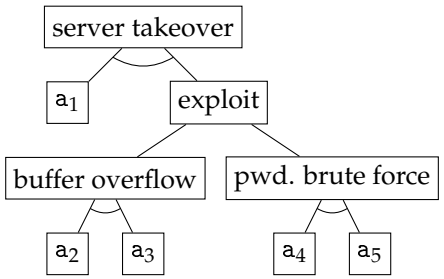
$$I(\mathtt{a}_2, T_1) = I(\mathtt{a}_3, T_1) = I(\mathtt{a}_4, T_1) = I(\mathtt{a}_5, T_1) = \tfrac{1}{2} \cdot \tfrac{1}{3} = \tfrac{1}{6} \,. \tag{C.6}$$

For attack tree T_2 we get:

$$I(\mathtt{a}_6, T_2) = I(\mathtt{a}_7, T_2) = I(\mathtt{a}_8, T_2) = \tfrac{1}{3} \,. \tag{C.7}$$

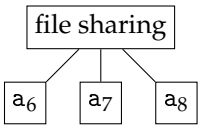
The overall damage of attack $\mathtt{a}_1$ can then be computed:

$$D(\mathtt{a}_1) = D(T_1) \cdot I(\mathtt{a}_1, T_1) + D(T_2) \cdot \overbrace{I(\mathtt{a}_1, T_2)}^{=0} = 900 \cdot \tfrac{1}{3} = 300 \,. \tag{C.8}$$



Attack	Description	Attack class
$\mathtt{a}_1$	horizontal scan	$\mathtt{A}_1$
$\mathtt{a}_2$	fingerprinting	$\mathtt{A}_2$
$\mathtt{a}_3$	buffer overflow	$\mathtt{A}_3$
$\mathtt{a}_4$	SSH pwd. bf. request	$\mathtt{A}_4$
$\mathtt{a}_5$	SSH pwd. bf. response	$\mathtt{A}_4$

Figure C.1: Example attack tree T_1.



Attack	Description	Attack class
$\mathtt{a}_6$	download	$\mathtt{A}_5$
$\mathtt{a}_7$	upload	$\mathtt{A}_5$
$\mathtt{a}_8$	directory node	$\mathtt{A}_5$

Figure C.2: Example attack tree T_2.

In the same way, we obtain for the other attacks:

$$D(\mathtt{a}_2) = D(\mathtt{a}_3) = D(\mathtt{a}_4) = D(\mathtt{a}_5) = 900 \cdot \tfrac{1}{6} = 150\,, \tag{C.9}$$

$$D(\mathtt{a}_6) = D(\mathtt{a}_7) = D(\mathtt{a}_8) = \tfrac{50}{3}\,. \tag{C.10}$$

Finally, we can compute the attack class damages:

$$D(\mathtt{A}_1) = D(\mathtt{A}_4) = 300\,, \tag{C.11}$$

$$D(\mathtt{A}_2) = D(\mathtt{A}_3) = 150\,, \tag{C.12}$$

$$D(\mathtt{A}_5) = 50\,. \tag{C.13}$$

So, for example, a fraction of $\frac{300}{950} = \frac{6}{19}$ of the challenges is chosen from attack class $\mathtt{A}_1$, and one out of 19 challenges comes from attack class $\mathtt{A}_5$.

Appendix D

Results of Trust Model Tuning

Table D.1 shows all computed expected utilities of the master against the optimal attack strategies of the attackers. These utilities have been computed for different fractions of attackers f_a among the workers. The last column represents the weighted average of the utilities; the weights 0.3, 0.3, 0.2 and 0.2 (in this order) have been applied. The six best combined utilities are marked. The results are shown in Figure D.1, D.2, D.3, D.4 and D.5.

Table D.1: Utilities of the master against optimal attack strategies.

B^+	s_T ϵ	W	$\mathcal{U}_T$ $f_a = 0$	$f_a = 0.1$	$f_a = 0.5$	$f_a = 0.9$	combined
0	0.1	W_1	9.6273	9.6495	9.8142	9.9647	9.7388
0.05	0.1	W_1	9.6271	9.6765	9.8027	9.9668	9.745
0.1	0.1	W_1	9.6272	9.6636	9.8103	9.9652	9.7424
0.25	0.1	W_1	9.6304	9.6605	9.7958	9.9662	9.7397
0.5	0.1	W_1	9.6288	9.6717	9.8185	9.9692	9.7477
0.75	0.1	W_1	9.6215	9.6749	9.8185	9.966	9.7458
1	0.1	W_1	9.6339	9.6758	9.8236	9.9676	9.7511
1.5	0.1	W_1	9.6309	9.6668	9.8171	9.968	9.7463
2	0.1	W_1	9.6258	9.6583	9.8298	9.9646	9.7441
5	0.1	W_1	9.6259	9.6736	9.8303	9.967	9.7493
0	0.25	W_1	9.7802	9.7942	9.7998	9.6154	9.7554
0.05	0.25	W_1	9.7854	9.8119	9.761	9.522	9.7358
0.1	0.25	W_1	9.7937	9.7932	9.8329	9.7251	9.7877
0.25	0.25	W_1	9.8245	9.82	9.9069	9.6816	9.811
0.5	0.25	W_1	9.8514	9.8626	9.9259	9.7899	9.8573
0.75	0.25	W_1	9.8707	9.8907	9.9325	9.8392	9.8828
1	0.25	W_1	9.8936	9.9387	9.9886	9.8828	9.9184
1.5	0.25	W_1	9.8927	9.9075	9.9466	9.8281	9.895
2	0.25	W_1	9.9005	9.9151	9.9546	9.7444	9.8845

continued on next page ...

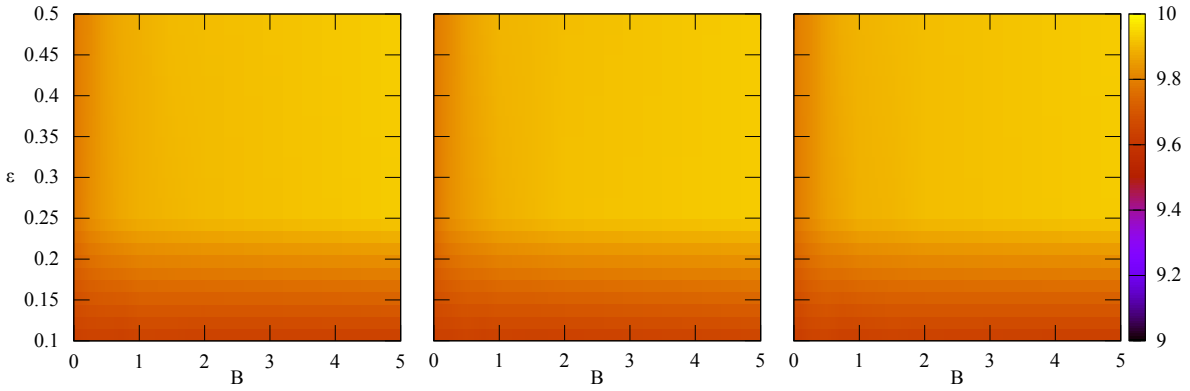
. . . continued from previous page

B^+	ϵ	W	$f_a = 0$	$f_a = 0.1$	$f_a = 0.5$	$f_a = 0.9$	combined
5	0.25	W_1	9.9278	9.9348	9.8001	9.7617	9.8711
0	0.5	W_1	9.7796	9.7879	9.6263	9.5988	9.7153
0.05	0.5	W_1	9.7815	9.7993	9.757	9.7396	9.7736
0.1	0.5	W_1	9.7973	9.8042	9.7989	9.7303	9.7863
0.25	0.5	W_1	9.8132	9.8315	9.9121	9.6993	9.8157
0.5	0.5	W_1	9.8557	9.8669	9.8029	9.0901	9.6954
0.75	0.5	W_1	9.875	9.8742	9.786	9.5225	9.7865
1	0.5	W_1	9.8818	9.8978	9.938	9.8197	9.8854
1.5	0.5	W_1	9.9009	9.9134	9.9493	8.961	9.7263
2	0.5	W_1	9.9038	9.9165	9.9517	9.2688	9.7902
5	0.5	W_1	9.9295	9.9119	9.4712	9.1308	9.6728
0	0.1	W_2	9.6303	9.6443	9.8142	9.9651	9.7382
0.05	0.1	W_2	9.6244	9.6499	9.8164	9.9609	9.7377
0.1	0.1	W_2	9.6244	9.6648	9.817	9.9585	9.7419
0.25	0.1	W_2	9.6238	9.6526	9.8068	9.9588	9.736
0.5	0.1	W_2	9.6328	9.6704	9.826	9.9657	9.7493
0.75	0.1	W_2	9.6225	9.6624	9.8229	9.9637	9.7428
1	0.1	W_2	9.6279	9.6787	9.8233	9.9657	9.7498
1.5	0.1	W_2	9.6226	9.6732	9.8291	9.9678	9.7481
2	0.1	W_2	9.6231	9.6657	9.8296	9.9672	9.746
5	0.1	W_2	9.6384	9.6672	9.8346	9.9709	9.7528
0	0.25	W_2	9.7734	9.7687	9.7537	9.5445	9.7223
0.05	0.25	W_2	9.7762	9.7991	9.7761	9.7686	9.7815
0.1	0.25	W_2	9.7955	9.8107	9.7581	9.5567	9.7448
0.25	0.25	W_2	9.8255	9.832	9.9134	9.6887	9.8177
0.5	0.25	W_2	9.8493	9.8758	9.9293	9.9262	9.8886
0.75	0.25	W_2	9.8668	9.8763	9.9162	9.9022	9.8866
1	0.25	W_2	9.8826	9.8889	9.9427	9.9869	9.9174
1.5	0.25	W_2	9.8953	9.9113	9.9495	9.8543	9.9027
2	0.25	W_2	9.9092	9.9192	9.9569	9.7481	9.8895
5	0.25	W_2	9.9285	9.9327	9.8144	9.736	9.8684
0	0.5	W_2	9.776	9.7934	9.7061	9.599	9.7319
0.05	0.5	W_2	9.7897	9.8055	9.6232	9.4801	9.6992
0.1	0.5	W_2	9.7963	9.8139	9.9063	9.4352	9.7514
0.25	0.5	W_2	9.8158	9.8473	9.7407	9.764	9.7999
0.5	0.5	W_2	9.8572	9.8773	9.9052	9.5854	9.8184
0.75	0.5	W_2	9.8762	9.8877	9.934	9.593	9.8346
1	0.5	W_2	9.8917	9.8935	9.9447	9.1202	9.7486
1.5	0.5	W_2	9.8966	9.9052	9.9489	8.9821	9.7267
2	0.5	W_2	9.906	9.9151	9.9581	9.3999	9.8179
5	0.5	W_2	9.9285	9.7894	9.4723	9.1369	9.6372
0	0.1	W_3	9.6196	9.6608	9.819	9.965	9.7409
0.05	0.1	W_3	9.626	9.6664	9.8095	9.9609	9.7418
0.1	0.1	W_3	9.6178	9.6529	9.8138	9.9636	9.7367
0.25	0.1	W_3	9.6335	9.6696	9.7981	9.9619	9.7429
0.5	0.1	W_3	9.6362	9.6626	9.8156	9.9652	9.7458

continued on next page . . .

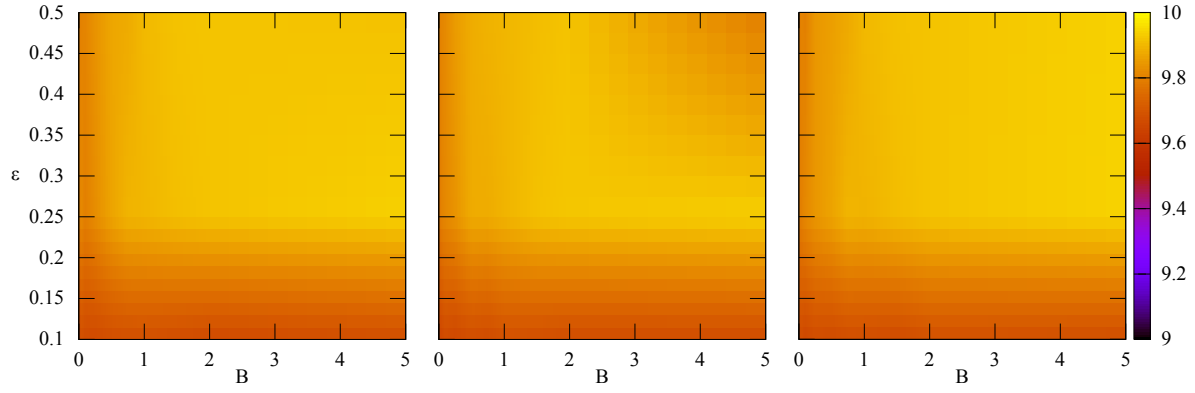
...continued from previous page

B^+	ϵ	W	$f_a = 0$	$f_a = 0.1$	$f_a = 0.5$	$f_a = 0.9$	combined
0.75	0.1	W_3	9.6201	9.6706	9.8243	9.9673	9.7455
1	0.1	W_3	9.623	9.6703	9.8297	9.9675	9.7474
1.5	0.1	W_3	9.6313	9.6611	9.8257	9.9664	9.7462
2	0.1	W_3	9.6256	9.6795	9.8347	9.9702	9.7525
5	0.1	W_3	9.6175	9.6726	9.8349	9.967	9.7474
0	0.25	W_3	9.7675	9.7835	9.7567	9.7322	9.7631
0.05	0.25	W_3	9.7819	9.796	9.8675	9.7293	9.7928
0.1	0.25	W_3	9.7943	9.8117	9.8917	9.7245	9.805
0.25	0.25	W_3	9.8183	9.8376	9.9108	9.7008	9.8191
0.5	0.25	W_3	9.8527	9.8617	9.9219	9.8523	9.8692
0.75	0.25	W_3	9.8686	9.8922	9.9373	9.9886	9.9134
1	0.25	W_3	9.8854	9.8882	9.9461	9.8737	9.896
1.5	0.25	W_3	9.891	9.9047	9.9493	9.8027	9.8891
2	0.25	W_3	9.9048	9.9156	9.9517	9.8457	9.9056
5	0.25	W_3	9.927	9.9386	9.9596	9.7985	9.9113
0	0.5	W_3	9.7849	9.7759	9.8075	9.6339	9.7565
0.05	0.5	W_3	9.7853	9.8015	9.6787	9.5842	9.7286
0.1	0.5	W_3	9.8	9.7927	9.8079	9.5677	9.7529
0.25	0.5	W_3	9.8154	9.8393	9.9063	9.5652	9.7907
0.5	0.5	W_3	9.8572	9.8537	9.9277	9.8215	9.8631
0.75	0.5	W_3	9.8804	9.8718	9.9383	9.4316	9.7997
1	0.5	W_3	9.8885	9.8977	9.9459	9.4899	9.823
1.5	0.5	W_3	9.8979	9.9112	9.8511	9.4554	9.804
2	0.5	W_3	9.9057	9.9144	9.9548	9.002	9.7374
5	0.5	W_3	9.9284	9.9362	9.5286	9.2062	9.7063



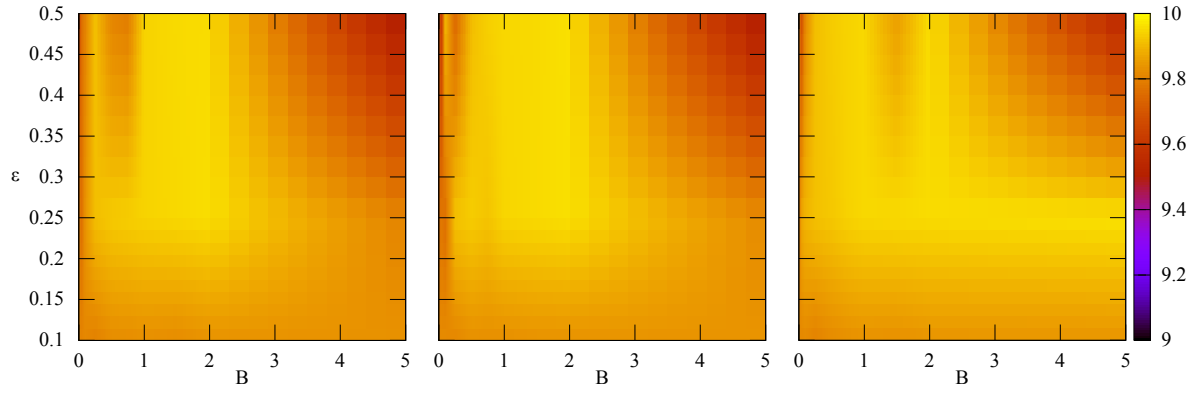
(a) W_1 (b) W_2 (c) W_3

Figure D.1: Expected utilities of the master against optimal attack strategies ($f_a = 0$).



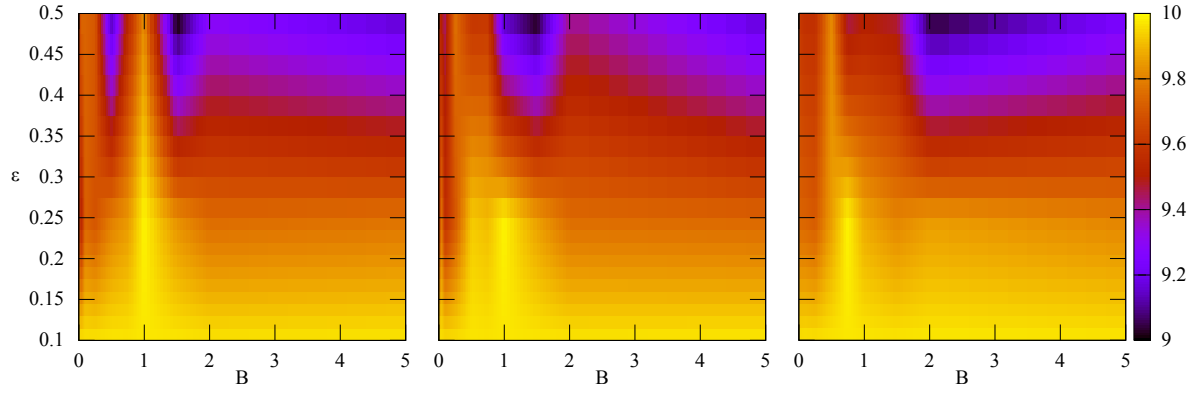
(a) W_1 (b) W_2 (c) W_3

Figure D.2: Expected utilities of the master against optimal attack strategies ($f_a = 0.1$).



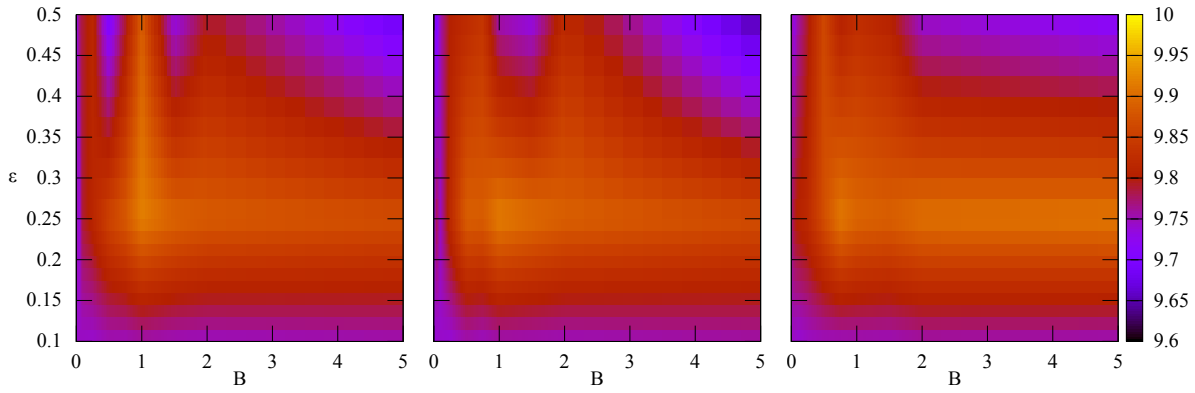
(a) W_1 (b) W_2 (c) W_3

Figure D.3: Expected utilities of the master against optimal attack strategies ($f_a = 0.5$).



(a) W_1 (b) W_2 (c) W_3

Figure D.4: Expected utilities of the master against optimal attack strategies ($f_a = 0.9$).



(a) W_1 (b) W_2 (c) W_3

Figure D.5: Expected utilities of the master combined for different scenarios.

Appendix E

Correlation of Sources

E.1 Computing Correlation

In this section, we show how the correlations between pairs of honest sources, colluders and mixed pairs can be computed. As argued in Section 5.4.1, we assume that the set of sources $\mathcal{S}$ is large, so that we can use sampling with replacement. First, it is shown how the correlations are computed for UCm and CCm attacker models respectively. Then, the resulting correlations are shown.

E.1.1 Case UCm Attackers

In this case, colluders decide to collude with a probability of p_{cam}, independent of how many colluders are in the vote.

E.1.1.1 Case h&h

See Section 5.4.1.

E.1.1.2 Case h&c

Given that they are in the same vote, a honest source A and a colluder B are in the same group (plurality/minority), iff any of the following is the case:

- both return the correct information: $P_1 = (1-\rho)\cdot(1-p_{\text{cam}})$.
- A returns incorrect information, and B colludes: $P_2 = \rho \cdot p_{\text{cam}}$, and the colluder is not in a plurality, i.e., there are more correct honest sources (j) than colluders (i) in the remaining vote:

$$P_3 = \sum_{i=0}^{\lceil \frac{k-3}{2} \rceil} \binom{k-2}{i} {f_{\text{a}}}^i \cdot \sum_{j=i+1}^{k-2-i} \binom{k-2-i}{j} {p_h}^j \cdot {p_w}^{k-2-i-j} \,. \tag{E.1}$$

- honest source A returns the correct information, and the colluders decide to collude: $P_4 = (1-\rho)\cdot p_{\text{cam}}$, and there is a tie in the vote, i.e., there are equally many correct honest sources as colluders

(in the remaining vote):

$$P_5 = \sum_{i=0}^{\lfloor \frac{k-2}{2} \rfloor} \binom{k-2}{i} {f_a}^i \cdot \binom{k-2-i}{i} {p_h}^i \cdot {p_w}^{k-2-2i} \tag{E.2}$$

$$= \sum_{i=0}^{\lfloor \frac{k-2}{2} \rfloor} \frac{k-2}{i!^2 \cdot (k-2-2i)!} \cdot {f_a}^i \cdot {p_h}^i \cdot {p_w}^{k-2-2i} , \tag{E.3}$$

which is a sum over the PMF of the *multinomial distribution*.

- honest source A returns incorrect information, and the colluders decide to not collude, and so return the correct information: $P_6 = \rho \cdot (1 - p_{\text{cam}})$, but at the same time there is no other colluder and no correct honest source (otherwise the colluder B would be in the plurality, and A not): $P_7 = {p_w}^{k-2}$.

Again, the cases are mutually exclusive, and we can add them up:

$$p_c^{UCm}(h\&c) = P_1 + P_2 \cdot P_3 + P_4 \cdot P_5 + P_6 \cdot P_7 . \tag{E.4}$$

E.1.1.3 Case c&c

Two colluders take collectively the same decision, so are always in the same group:

$$p_c^{UCm}(c\&c) = 1 . \tag{E.5}$$

E.1.2 Case CCm Attackers

In this case, colluders decide to collude with a probability of p_{cam}, but only in votes they are sure to win, that is where there are at least $\left\lceil \frac{k+1}{2} \right\rceil$ many colluders.

E.1.2.1 Case h&h

Given that they are in the same vote, two honest sources A and B are also together in the same group (plurality/minority), iff any of the following is the case:

- they both return the correct information: $P_1 = (1 - \rho)^2$.
- they both return incorrect information: $P_2 = \rho^2$.
- exactly one of them returns the correct information: $P_3 = 2 \cdot \rho \cdot (1 - \rho)$, and,
  - apart from them there are only incorrect honest sources: $P_4 = {p_w}^{k-2}$, or,
  - there are at least $\left\lceil \frac{k+1}{2} \right\rceil$ colluders and these decide to collude:

$$P_5 = p_{\text{cam}} \cdot \sum_{i=\lceil \frac{k+1}{2} \rceil}^{k-2} \binom{k-2}{i} {f_a}^i \cdot (1 - f_a)^{k-2-i} . \tag{E.6}$$

Because these cases are mutually exclusive, the correlation is given by:

$$p_c^{CCm}(h\&h) = P_1 + P_2 + P_3 \cdot (P_4 + P_5) . \tag{E.7}$$

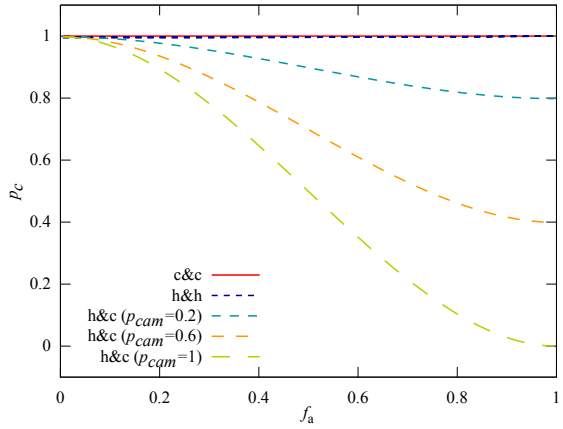


Figure E.1: Correlations between different types of sources (CCm, $k = 5$, $\rho = 0.00221$).

E.1.2.2 Case h&c

Given that they are in the same vote, a honest source A and a colluder B are together in the (plurality/minority), iff any of the following is the case:

- A returns the correct information: $P_1 = (1 - \rho)$, and the colluders also return the correct information, which means that either:
  - the colluders are not in the majority (note that B is counted in):

$$P_2 = \sum_{i=0}^{\lfloor \frac{k-2}{2} \rfloor} \binom{k-2}{i} {f_a}^i \cdot (1 - f_a)^{k-2-i} , \tag{E.8}$$

  - or they are in the majority, but decide to not collude: $P_3 = (1 - P_2) \cdot (1 - p_{\text{cam}})$.

Because a honest source can only be in the plurality if it returns the correct information, and a conditional colluder is never in the minority, we do not have to consider the cases in which A returns incorrect information. The above cases can be combined as follows:

$$p_c^{UCm}(h\&c) = P_1 \cdot P_2 + P_1 \cdot P_3 . \tag{E.9}$$

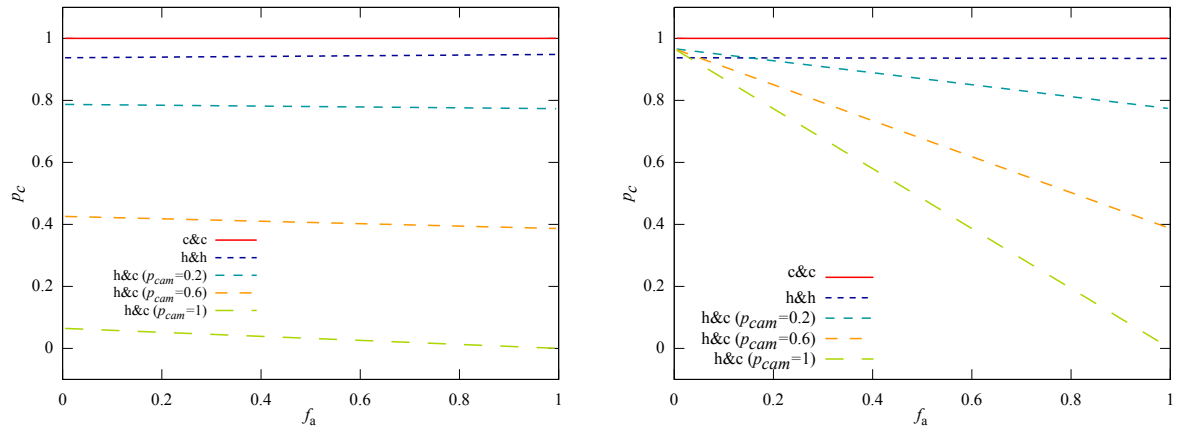
E.1.2.3 Case c&c

Two colluders take collectively the same decision, so are always in the same group:

$$p_c^{CCm}(c\&c) = 1 . \tag{E.10}$$

E.1.3 Figures

In this section, we show the correlations of cases, which have not been shown in Section 5.4.2, but give interesting insights. Figure E.1 shows the correlations for CCm attackers when a higher redundancy level $k = 5$ is used. The difference between the correlation of h&h and h&c is smaller compared to the



(a) UCm (b) CCm

Figure E.2: Correlations between different types of sources ($k = 3$, $\rho = 0.0335$).

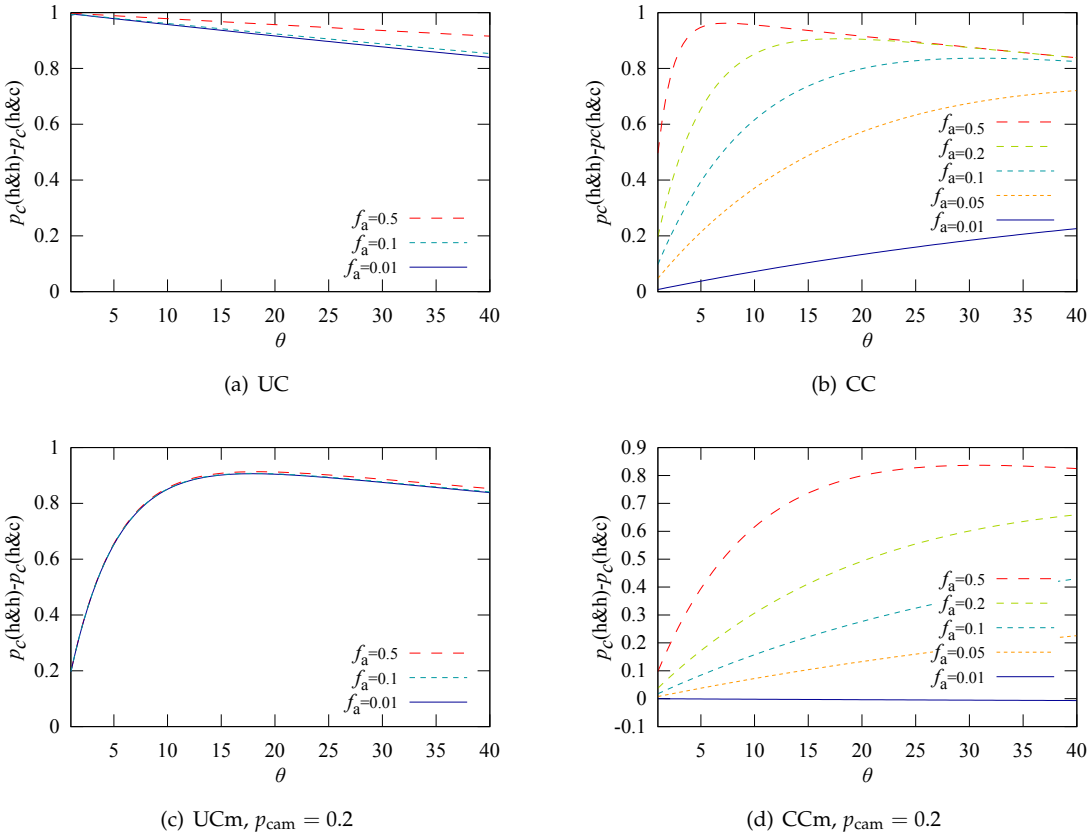
case of $k = 3$ (compare to Fig. 5.5 in Sect. 5.4.2). The reason is that for higher redundancy levels, the conditional colluders are for smaller f_a less often in the majority. As a consequence, they collude less often and so correlate more with honest sources. This makes them again less harmful but also harder to detect. The correlations for UCm and $k = 5$ are not shown because they are similar to the case of $k = 3$ (see Fig. 5.5 in Sect. 5.4.2).

Figure E.2 shows correlations for scenarios in which honest sources would have high error rates ($\rho = 0.0335$ with respect to [88]). Both in the case of UCm and CCm, h&h is lower, that is the honest sources correlate less with each other. Therefore the differences between h&c and h&h become smaller, which makes it again harder to find the borderline between colluders and honest sources for small f_a. Still, depending on the camouflage probability p_{cam}, the difference in the UCm case is still big.

E.2 Amplification

Pairs consisting of a honest source and a colluder in most cases correlate less – both for UC and CC – than pairs consisting only of colluders *or* honest sources (see Sect. 5.4.2). Put simply, if this difference is big, it is easier to detect colluders. In the CC case, the difference becomes smaller for small ratios of colluders f_a. In order to increase this difference for the clustering algorithm, the correlation can be amplified by exponentiation with some parameter θ. For $a, b \in (0, 1)$ with $a > b$ and $\theta > 1$, it holds: $\frac{a}{b} < \frac{a^\theta}{b^\theta}$. This means that through exponentiation, higher correlations become relatively even higher compared to lower correlations. If the used clustering algorithm interprets relations between edge weights in a relative manner (e.g., probabilistically), exponentiation can generally be used as amplifier. When the relations are interpreted as absolute differences, exponentiation can help as well, but can make the situation also worse, as we will discuss in the following.

The difference between two correlation values a and b, with $a > b$, decreases in some cases for increasing exponentiation. This is shown in Figure E.3 where the difference for the correlations of h&h and h&c pairs is depicted when applying different θ (the figure for $p_c(c\&c) - p_c(h\&c)$ would look similar). As mentioned in Section 5.4.2, high differences in correlation of h&c and h&h/c&c pairs mean



(a) UC

(b) CC

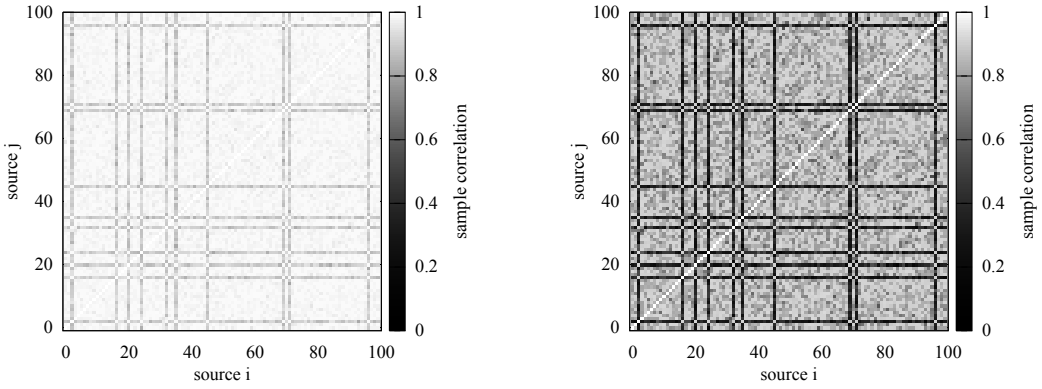
(c) UCm, $p_{\text{cam}} = 0.2$

(d) CCm, $p_{\text{cam}} = 0.2$

Figure E.3: Effect of amplification on $p_c(h\&h) - p_c(h\&c)$.

good identifiability. So, a lower difference makes colluders harder to identify, which is not desirable. While the amplification helps in the CC case for θ up to about 20, it makes it worse in the UC case (for θ higher than ca. 1.4). Still, the differences in the UC case remain high. Hence, a choice of $\theta = 20$ for instance makes it much easier to spot h&c pairs in the CC case but not much harder in the generally easier UC case. For camouflage colluders with a low camouflage rate, the amplification helps also in the unconditional case (Fig. E.3(c)). For the case of very few conditional camouflage colluders (Fig. E.3(d), $f_a = 0.01$), the amplification helps the colluders and makes the difference worse; however, in this case colluders cannot be recognized anyway. Finally, we have to keep in mind that in our algorithm, we are making an estimate of the actual correlation. Errors in estimates are higher for fewer observations, and so, the amplification has to be used carefully when there are few observations, because it amplifies the errors in the estimates as well.

The amplification can be carried out by element-wise exponentiation of the correlation matrix with scalar θ. Figure E.4 shows the correlation matrices for the same H_S, before and after amplification ($\theta = 10$). This illustrates that amplification makes h&c pairs better discernible.



(a) $\theta = 1$ (b) $\theta = 10$

Figure E.4: Correlation matrix based on 100 votes/pair for CC case ($|\mathcal{S}| = 100$).

Nomenclature

Introduction

$\mathcal{S}$	Set of all available information sources	4
$\mathcal{F}$	Set of existing facts	2
$\mathcal{R}$	A single response	2
$\mathcal{Q}$	A single request	2
$v(x)$	Explicit verification function	2

Chapter 3

$\mathtt{A}$	Attack class	33
α	Attack score for malicious flow	30
$\mathtt{a}$	A specific network attack	34
$\bar{\lambda}$	Sample mean for attack scores of legitimate flows	31
$\bar{\alpha}$	Sample mean for attack scores of malicious flows	31
CDF_y	Cumulative distribution function for given y	22
$\mathcal{C}$	Confidence for a margin of error	31
$\mathcal{K}$	Set of known facts	20
λ	Attack score for legitimate flow	31
$\mathbb{E}$	Expected value	23
Δ_L	Left margin of error	32
Δ_R	Right margin of error	32
k	Redundancy level: A request is replicated k times	28
s_λ	Sample standard deviation for attack scores of legitimate flows	31
s_α	Sample standard deviation for attack scores of malicious flows	31
a_i	Aggregator i	32
C_i	ith clause of a propositional logic formula in DNF (as set)	35
D	Expected damage of realized threats or single attacks	33
$F(T)$	An attack tree in minimal DNF represented as set of clauses	35
$I(\mathtt{a}, T)$	Importance of attack $\mathtt{a}$ in attack tree T	36
i_max	Maximum number of incorrect replies to M that are tolerated	26
k_max	Only the topmost k_max aggregators are considered	33
M	Set of real requests	20

m	Number of real requests	21
N	Set of challenges	20
n	Number of challenges	21
r	Overall rating of an aggregator	31
r_i	Rating of an aggregator for batch i	31
T	Attack tree	34
X	Random variable for the number of incorrect replies to M	21
Y	Random variable for the number of incorrect replies to N	20
Z	Random variable for the number of incorrect replies to $M \cup N$	20
CDF	Cumulative distribution function	22
DNF	Disjunctive Normal Form of a propositional logic formula	35
PMF	Probability mass function	21

Chapter 4

ψ	An action performed by a malicious entity.	66
Ψ	Set of all possible actions ψ of a malicious entity.	66
A	Aging factor for the computation of a trust value	51
$\text{br}^*(s_T)$	Best response to trustor's strategy s_T that minimizes utility of trustor	68
ρ	Probability with which a source returns incorrect reply to a single request	49
f_a	Fraction of malicious workers among all workers.	71
τ	A service requested by the trustor.	66
ϵ	Tolerance factor that defines when evidence should be combined	52
t	Trust value as weighted average.	51
t_i	Trust value as exponential moving average, after i replies from a source.	51
$\dot{t}$	Trust value with the idea of combined evidence.	52
$\dot{t}^{W_3}$	estimate $1 - \dot{t}$, where $\dot{t}$ is computed with weight $(w_i)_i = (1)$	59
$\mathcal{U}_A$	Expected utility of an attacker for an entire game.	67
$\mathcal{U}_T$	Expected utility of a trustor for an entire game.	67
$\mathtt{U}_A$	Utility of an attacker for a single round.	67
$\mathtt{U}_T$	Utility of a trustor for a single round.	67
S_A	Set of all possible attack strategies, i.e., strategies of malicious entities.	67
S_T	Set of all possible trust model configurations.	66
w_i	Weight used for the computation of a trust value.	51

Chapter 5

$\mathcal{S}$	Set of all available information sources	79
f_a	Fraction of colluding sources within $\mathcal{S}$.	78
m_k	Number of sources needed to form a majority for redundancy level k	78
k	Redundancy level: to how many sources a request is made redundantly	78

θ	Amplification exponent	128
$H_{\mathcal{S}}$	Voting history of sources in $\mathcal{S}$	82
CC	Conditional collusion strategy	80
CCm	Conditional camouflage collusion strategy	80
p_c	Correlation in the voting history of two sources	81
UC	Unconditional collusion strategy	80
UCm	Unconditional camouflage collusion strategy	80

List of Figures

List of Tables

Author's Publications

[FsS'08] Volker Fusenig, Eugen Staab, Uli Sorger, and Thomas Engel. Unlinkable communication. In Larry Korba, Steve Marsh, and Reihaneh Safavi-Naini, editors, *Proc. of the 6th Annual Conf. on Privacy, Security and Trust (PST '08)*, pages 51–Ű55. IEEE Computer Society, October 2008.

[FsS'09] Volker Fusenig, Eugen Staab, Uli Sorger, and Thomas Engel. Slotted packet counting attacks on anonymity protocols. In Ljiljana Brankovic and Willy Susilo, editors, *Proc. of the Australasian Information Security Conference (AISC '09)*, volume 98 of *CRPIT Series*, pages 53–59. Australian Computer Society, 2009.

[RsF'09a] Martin Rehák, Eugen Staab, Volker Fusenig, Michal Pěchouček, Martin Grill, Jan Stiborek, Karel Bartoš, and Thomas Engel. Runtime monitoring and dynamic reconfiguration for intrusion detection systems. In E. Kirda, S. Jha, and D. Balzarotti, editors, *Recent Advances in Intrusion Detection: Proc. of the 12th Int. Symposium On Recent Advances In Intrusion Detection (RAID '09)*, volume 5758 of *LNCS*, pages 61–80. Springer-Verlag, September 2009.

[RsF'09b] Martin Rehák, Eugen Staab, Volker Fusenig, Jan Stiborek, Martin Grill, Karel Bartoš, Michal Pěchouček, and Thomas Engel. Threat-model driven runtime adaptation and evaluation of intrusion detection system. In *Proc. of the 6th Int. Conf. on Autonomic Computing and Communications (ICAC '09)*, pages 65–66. ACM, June 2009.

[RsP'09] Martin Rehák, Eugen Staab, Michal Pěchouček, Jan Stiborek, Martin Grill, and Karel Bartoš. Dynamic information source selection for intrusion detection systems. In K. S. Decker, J. S. Sichman, C. Sierra, and C. Castelfranchi, editors, *Proc. of the 8th Int. Conf. on Autonomous Agents & Multiagent Systems (AAMAS '09)*, pages 1009–1016. IFAAMAS, May 2009.

[sC'09a] Eugen Staab and Martin Caminada. Assessing the impact of informedness on a consultant's profit. Technical report, University of Luxembourg, September 2009. `http://arxiv.org/abs/0909.0901/`. [Online; accessed 2010-02-08].

[sC'09b] Eugen Staab and Martin Caminada. Simulating knowledge and dishonesty in a client-consultant setting. In Toon Calders, Karl Tuyls, and Mykola Pechenizkiy, editors, *Proc. of the 21st Benelux Conf. on Artificial Intelligence (BNAIC '09)*, pages 397–398, October 2009.

[sC'10] Eugen Staab and Martin Caminada. On the profitability of incompetence. In *Proc. of the 11th Workshop on Multi-Agent-Based Simulation (MABS '10)*, 2010.

[sE'07] Eugen Staab and Thomas Engel. Formalizing excusableness of failures in multi-agent systems. In Aditya Ghose, Guido Governatori, and Ramakoti Sadananda, editors, *Agent Computing and Multi-Agent Systems: Revised Papers of the 10th Pacific Rim Int. Conf. on Multi-Agents (PRIMA '07)*, volume 5044 of *LNCS (LNAI)*, pages 122–133. Springer-Verlag, November 2009.

[sE'08a] Eugen Staab and Thomas Engel. Combining cognitive and computational concepts for experience-based trust reasoning. In Rino Falcone, Suzanne Barber, Jordi Sabater, and Munindar Singh, editors, *Proc. of the 11th Int. Workshop on Trust in Agent Societies (TRUST '08)*, pages 41–45. IFAAMAS, May 2008.

[sE'08b] Eugen Staab and Thomas Engel. Combining cognitive with computational trust reasoning. In Rino Falcone, Suzanne Barber, Jordi Sabater, and Munindar Singh, editors, *Trust in Agent Societies: Revised Selected and Invited Papers of the 11th Int. Workshop on Trust in Agent Societies (TRUST '08)*, volume 5396 of *LNCS (LNAI)*, pages 99–111. Springer-Verlag, December 2008.

[sE'09a] Eugen Staab and Thomas Engel. Collusion detection for distributed computing. In Sjouke Mauw and Leendert van der Torre, editors, *Proc. of the 1st Luxembourg Day on Security and Reliability (P1Day '09)*, page 75. University of Luxembourg, February 2009.

[sE'09b] Eugen Staab and Thomas Engel. Collusion detection for grid computing. In Franck Cappello, Cho-Li Wang, and Rajkumar Buyya, editors, *Proc. of the 9th IEEE/ACM Int. Symposium on Cluster Computing and the Grid (CCGrid '09)*, pages 412–419. IEEE Computer Society, May 2009.

[sE'09c] Eugen Staab and Thomas Engel. Tuning evidence-based trust models. In *Proc. of the 2009 IEEE Int. Conf. on Information Privacy, Security, Risk and Trust (PASSAT '09)*, pages 92–99. IEEE Computer Society, August 2009.

[sFE'08a] Eugen Staab, Volker Fusenig, and Thomas Engel. Towards trust-based acquisition of unverifiable information. In Matthias Klusch, Michal Pěchouček, and Axel Polleres, editors, *Cooperative Information Agents XII: Proc. of the 12th Int. Workshop on Cooperative Information Agents (CIA '08)*, volume 5180 of *LNCS (LNAI)*, pages 41–54. Springer-Verlag, September 2008.

[sFE'08b] Eugen Staab, Volker Fusenig, and Thomas Engel. Using correlation for collusion detection in grid settings. Technical Report 000657499, University of Luxembourg, July 2008.

[sFE'08c] Eugen Staab, Volker Fusenig, and Thomas Engel. Trust-aided acquisition of unverifiable information. In Malik Ghallab, Constantine D. Spyropoulos, Nikos Fakotakis, and Nikolaos M. Avouris, editors, *Proc. of the 18th European Conf. on Artificial Intelligence (ECAI '08)*, volume 178 of *Frontiers in Artificial Intelligence and Applications*, pages 869–870. IOS Press, July 2008.

Bibliography

[1] distributed.net. http://www.distributed.net/. [Online; accessed 2010-01-25].

[2] Folding@home. http://folding.stanford.edu/. [Online; accessed 2010-01-25].

[3] Great Internet Mersenne Prime Search (GIMPS). http://www.mersenne.org/. [Online; accessed 2010-01-25].

[4] LHC@home. http://lhcathome.cern.ch/. [Online; accessed 2010-01-25].

[5] W3c web services glossary. http://www.w3.org/TR/ws-gloss/. [Online; accessed 2010-01-25].

[6] Wikipedia, The Free Encyclopedia. http://wikipedia.org/. [Online; accessed 2010-01-25].

[7] Alfarez Abdul-Rahman. The PGP trust model. EDI-Forum, 1997.

[8] Ian F. Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38(4):393–422, 2002.

[9] Carlos E. Alchourron, Peter Gärdenfors, and David Makinson. On the logic of theory change: Partial meet functions for contraction and revision. *Journal of Symbolic Logic*, 50:510–530, 1985.

[10] David P. Anderson. BOINC: A system for public-resource computing and storage. In *Proc. of the 5th IEEE/ACM Int. Workshop on Grid Computing (GRID '04)*, pages 4–10. IEEE Computer Society, 2004.

[11] David P. Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky, and Dan Werthimer. SETI@home: an experiment in public-resource computing. *Communications of the ACM*, 45(11):56–61, 2002.

[12] Donovan Artz and Yolanda Gil. A survey of trust in computer science and the semantic web. *Web Semant.*, 5(2):58–71, 2007.

[13] Venkatesan Balakrishnan, Vijay Varadharajan, Phillip Lucs, and Udaya Kiran Tupakula. Trust enhanced secure mobile ad-hoc network routing. In *Proc. of the 21st Int. Conf. on Advanced Information Networking and Applications (AINA '07), Workshop Proceedings Vol. 2*, pages 27–33. IEEE Computer Society, 2007.

[14] Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation clustering. *Machine Learning*, 56(1–3):89–113, 2004.

[15] Arash Baratloo, Mehmet Karaul, Zvi M. Kedem, and P. Wijckoff. Charlotte: Metacomputing on the web. *Future Generation Comp. Syst.*, 15(5-6):559–570, 1999.

[16] K. Suzanne Barber and Joonoo Kim. Belief revision process based on trust: Agents evaluating reputation of information sources. In *Trust in Cyber-societies: Integrating the Human and Artificial Perspectives*, volume 2246 of *LNCS*, pages 73–82. Springer-Verlag, 2001.

[17] Thomas Bayes. An essay toward solving a problem in the doctrine of chances. *Philosophical Transactions of the Royal Society of London*, 53:370–418, 1763.

[18] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, 2001.

[19] Elisa Bertino, Chenyun Dai, and Murat Kantarcioglu. The challenge of assuring data trustworthiness. In *Database Systems for Advanced Applications: Proc. of the 14th Int. DASFAA Conference*, volume 5463 of *LNCS*, pages 22–33. Springer-Verlag, 2009.

[20] Ulrik Brandes, Marco Gaertler, and Dorothea Wagner. Experiments on graph clustering algorithms. In *Algorithms: 11th Annual ESA Symposium*, volume 2832 of *LNCS*, pages 568–579. Springer, 2003.

[21] Leo Breiman. Bagging predictors. *Mach. Learn.*, 24(2):123–140, 1996.

[22] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks*, 30(1-7):107–117, 1998.

[23] Sonja Buchegger and Jean-Yves Le Boudec. A robust reputation system for mobile ad hoc networks. Technical Report IC/2003/50, EPFL-IC-LCA, CH-1015 Lausanne, 2003.

[24] Levente Buttyán, Péter Schaffer, and István Vajda. RANBAR: RANSAC-based resilient aggregation in sensor networks. In *Proc. of the 4th ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN '06)*, pages 83–90. ACM, 2006.

[25] Levente Buttyán, Péter Schaffer, and István Vajda. Cora: Correlation-based resilient aggregation in sensor networks. *Ad Hoc Networks*, 7(6):1035–1050, 2009.

[26] Vinny Cahill, Elizabeth Gray, Jean-Marc Seigneur, Christian D. Jensen, Yong Chen, Brian Shand, Nathan Dimmock, Andy Twigg, Jean Bacon, Colin English, Waleed Wagealla, Sotirios Terzis, Paddy Nixon, Giovanna di Marzo Serugendo, Ciaran Bryce, Marco Carbone, Karl Krukow, and Mogens Nielsen. Using trust for secure collaboration in uncertain environments. *IEEE Pervasive Computing*, 02(3):52–61, 2003.

[27] Martin Caminada. Truth, lies and bullshit; distinguishing classes of dishonesty. In *Proc. of the 1st IJCAI Workshop on Social Simulation (SS@IJCAI)*, 2009.

[28] Licia Capra and Mirco Musolesi. Autonomic trust prediction for pervasive systems. In *Proc. of the 20th Int. Conf. on Advanced Information Networking and Applications (AINA '06), Vol. 2*, pages 481–488. IEEE Computer Society, 2006.

[29] Cristiano Castelfranchi and Rino Falcone. Principles of trust for MAS: Cognitive anatomy, social importance, and quantification. In *Proc. of the 3rd Int. Conf. on Multi-Agent Systems (ICMAS '98)*, pages 72–79. IEEE Computer Society, 1998.

[30] Cristiano Castelfranchi and Rino Falcone. Trust is much more than subjective probability: Mental components and sources of trust. In *Proc. of the 33rd Hawaii Int. Conf. on System Sciences (HICSS '00)*. IEEE Computer Society, 2000.

[31] Richard Chow, Philippe Golle, Markus Jakobsson, Lusha Wang, and XiaoFeng Wang. Making CAPTCHAs clickable. In *Proc. of the 9th Workshop on Mobile Computing Systems and Applications (HotMobile '08)*, pages 91–94. ACM, 2008.

[32] Murat Şensoy and Pinar Yolum. Ontology-based service representation and selection. *IEEE Trans. Knowl. Data Eng.*, 19(8):1102–1115, 2007.

[33] Chenyun Dai, Dan Lin, Elisa Bertino, and Murat Kantarcioglu. An approach to evaluate data trustworthiness based on data provenance. In *Secure Data Management: Proc. of the 5th VLDB SDM Workshop*, volume 5159 of *LNCS*, pages 82–98. Springer-Verlag, 2008.

[34] Mehdi Dastani, Andreas Herzig, Joris Hulstijn, and Leendert van der Torre. Inferring trust. In J. Leite and P. Torroni, editors, *Computational Logic in Multi-Agent Systems: Proc. of the 5th Int. CLIMA Workshop*, volume 3487 of *LNCS (LNAI)*, pages 144–160. Springer-Verlag, 2005.

[35] Erik D. Demaine, Dotan Emanuel, Amos Fiat, and Nicole Immorlica. Correlation clustering in general weighted graphs. *Theor. Comput. Sci.*, 361(2):172–187, 2006.

[36] Robert Demolombe. To trust information sources: a proposal for a modal logical framework. In C. Castelfranchi and Y-H. Tan, editors, *Trust and Deception in Virtual Societies*, pages 111–124. Kluwer Academic Publishers, 2001.

[37] Jay L. Devore. *Probability and Statistics for Engineering and the Sciences, Enhanced Review Edition*. Duxbury Press, seventh edition, 2008.

[38] Anind K. Dey. *Providing Architectural Support for Building Context-Aware Application*. PhD thesis, College of Computing, Georgia Institute of Technology, 2000.

[39] Renata Dividino, Simon Schenk, Sergej Sizov, and Steffen Staab. Provenance, trust, explanations - and all that other meta knowledge. *Künstliche Intelligenz*, 2009.

[40] John R. Douceur. The Sybil attack. In *Revised Papers from the 1st Int. Workshop on Peer-to-Peer Systems (IPTPS '02)*, pages 251–260. Springer-Verlag, 2002.

[41] Wenliang Du, Jing Jia, Manish Mangal, and Mummoorthy Murugesan. Uncheatable grid computing. In *Proc. of the 24th Int. Conf. on Distributed Computing Systems (ICDCS'04)*, pages 4–11. IEEE Computer Society, 2004.

[42] Claudiu Duma, Martin Karresand, Nahid Shahmehri, and Germano Caronni. A trust-aware, P2P-based overlay for intrusion detection. In *3rd Int. Workshop on P2P Data Management, Security and Trust (PDMST '06)*, pages 692–697. IEEE Computer Society, 2006.

[43] Susan T. Dumais, George W. Furnas, Thomas K. Landauer, Scott Deerwester, and Richard Harshman. Using latent semantic analysis to improve access to textual information. In *Proc. of the SIGCHI Conf. on Human Factors in Computing Systems (CHI '88)*, pages 281–285. ACM, 1988.

[44] Levent Ertöz, Eric Eilertson, Aleksandar Lazarevic, Pang-Ning Tan, Vipin Kumar, Jaideep Srivastava, and Paul Dokas. The MINDS - Minnesota Intrusion Detection System. In *Next Generation Data Mining*. MIT Press, 2004.

[45] Babak Esfandiari and Sanjay Chandrasekharan. On how agents make friends: Mechanisms for trust acquisition. In *Proc. of the 4th Workshop on Deception, Fraud and Trust in Agent Societies (at AAMAS '01)*, pages 27–34, 2001.

[46] Rino Falcone and Cristiano Castelfranchi. Social trust: a cognitive approach. *Trust and deception in virtual societies*, pages 55–90, 2001.

[47] Gary William Flake, Robert Endre Tarjan, and Kostas Tsioutsiouliklis. Graph clustering and minimum cut trees. *Internet Mathematics*, 1(4), 2004.

[48] Ian Foster and Carl Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, 1998.

[49] Harry G. Frankfurt. *On Bullshit*. Princeton University Press, 2005.

[50] Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In *Proc. of the 13th Int. Conf. on Machine Learning (ICML '96)*, pages 148–156, 1996.

[51] Batya Friedman, Peter H. Kahn Jr., and Daniel C. Howe. Trust online. *Commun. of the ACM*, 43(12):34–40, 2000.

[52] Nir Friedman and Joseph Y. Halpern. Belief revision: A critique. *J. of Logic, Lang. and Inf.*, 8(4), 1999.

[53] Drew Fudenberg and Jean Tirole. *Game theory*. MIT Press, Cambridge, MA, 1991.

[54] Karen 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 *4th Int. Joint Conf. on Autonomous Agents & Multiagent Systems (AAMAS '05)*, pages 512–518, 2005.

[55] Karen K. Fullam, Jisun Park, and K. Suzanne Barber. Trust-driven information acquisition for secure and quality decision-making. In *Proc. of Int. Conf. on Integration of Knowledge Intensive Multi-Agent Systems (KIMAS '05)*, pages 303–310, 2005.

[56] Carol J. Fung, Olga Baysal, Jie Zhang, Issam Aib, and Raouf Boutaba. Trust management for host-based collaborative intrusion detection. In *Managing Large-Scale Service Deployment: Proc. of the 19th IFIP/IEEE Int. DSOM Workshop*, volume 5273 of *LNCS*, pages 109–122. Springer Verlag, 2008.

[57] Carol J. Fung, Jie Zhang, Issam Aib, and Raouf Boutaba. Robust and scalable trust management for collaborative intrusion detection. In *Proc. of the 11th IFIP/IEEE Int. Symp. on Integrated Network Management (IM '09)*, pages 33–40. IEEE Computer Society, 2009.

[58] Dov Gabbay, Gabriella Pigozzi, and Odinaldo Rodrigues. Belief revision, belief merging and voting. In *Proc. of the 7th Conf. on Logic and the Foundations of Games and Decision Theory (LOFT '06)*, pages 71–78, 2006.

[59] Diego Gambetta. Can we trust trust? In Diego Gambetta, editor, *Trust: Making and Breaking Cooperative Relations*, chapter 13, pages 213–237. Basil Blackwell, 1988. Reprinted in electronic edition from Department of Sociology, University of Oxford.

[60] Cécile Germain-Renaud and Dephine Monnier-Ragaigne. Grid result checking. In *Proc. of the 2nd Conf. on Computing Frontiers (CF '05)*, pages 87–96. ACM, 2005.

[61] Yolanda Gil and Donovan Artz. Towards content trust of web resources. In *Proc. of the 15th Int. Conf. on World Wide Web (WWW '06)*, pages 565–574. ACM, 2006.

[62] Andrew V. Goldberg and Kostas Tsioutsiouliklis. Cut tree algorithms: An experimental study. *Journal of Algorithms*, 38(1):51–83, 2001.

[63] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.

[64] Philippe Golle. Machine learning attacks against the Asirra CAPTCHA. In *Proc. of the 15th ACM Conf. on Computer and Communications Security (CCS '08)*, pages 535–542. ACM, 2008.

[65] Philippe Golle and Stuart G. Stubblebine. Secure distributed computing in a commercial environment. In *Proc. of the 5th Int. Conf. on Financial Cryptography (FC '01)*, pages 289–304. Springer-Verlag, 2002.

[66] R. E. Gomory and T. C. Hu. Multi-terminal network flows. *Journal of SIAM*, 9:551–570, 1961.

[67] Daniel Lombraña González, Francisco Fernández de Vega, and Henri Casanova. Characterizing fault tolerance in genetic programming. In *Proc. of the 2009 Workshop on Bio-Inspired Algorithms for Distributed Systems (BADS '09)*, pages 1–10, 2009.

[68] Tyrone Grandison and Morris Sloman. Trust management tools for internet applications. In *Proc. of the 1st Int. Conf. on Trust Management (iTrust '03)*, pages 91–107. Springer-Verlag, 2003.

[69] Sven Ove Hansson. A survey of non-prioritized belief revision. *Erkenntnis*, 50:413–427, 1999.

[70] Morten Hertzum, Hans H. K. Andersen, Verner Andersen, and Camilla B. Hansen. Trust in information sources: seeking information from people, documents, and virtual agents. *Interacting with Computers*, 14(5):575–599, 2002.

[71] Kevin Hoffman, David Zage, and Cristina Nita-Rotaru. A survey of attack and defense techniques for reputation systems. *ACM Computing Surveys*, 41, 2009.

[72] Thomas Hofmann. Probabilistic latent semantic indexing. In *Proc. of the 22nd annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval (SIGIR '99)*, pages 50–57. ACM, 1999.

[73] John H. Holland. *Adaptation in natural and artificial systems*. MIT Press, Cambridge, MA, USA, 1992.

[74] Trung Dong Huynh, Nicholas R. Jennings, and Nigel R. Shadbolt. An integrated trust and reputation model for open multi-agent systems. *Auton. Agents Multi-Agent Syst.*, 13(2):119–154, 2006.

[75] Mihaela Ion, Andrea Danzi, Hristo Koshutanski, and Luigi Telesca. A peer-to-peer multidimensional trust model for digital ecosystems. In *Proc. of the 2nd IEEE Int. Conf. on Digital Ecosystems and Technologies*, pages 461–469, 2008.

[76] Ramaprabhu Janakiraman, Marcel Waldvogel, and Qi Zhang. Indra: A peer-to-peer approach to network intrusion detection and prevention. In *Proc. of the 12th IEEE Int. Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE '03), Workshop on Enterprise Security (ES)*, pages 226–231. IEEE Computer Society, 2003.

[77] Edwin T. Jaynes. *Probability Theory: The Logic of Science*. Cambridge University Press, 2003.

[78] Stephen C. Johnson. Hierarchical clustering schemes. *Psychometrika*, 32:241–254, 1967.

[79] Audun Jøsang and Roslan Ismail. The beta reputation system. In *Proc. of the 15th Bled Conf. on Electronic Commerce*, pages 324–337, 2002.

[80] Radu Jurca and Boi Faltings. Mechanisms for making crowds truthful. *J. Artif. Intell. Res.*, 34:209–253, 2009.

[81] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The EigenTrust algorithm for reputation management in P2P networks. In *Proc. of the 12th Int. World Wide Web Conference (WWW '03)*, pages 640–651. ACM, 2003.

[82] George Karypis and Euihong Han. Concept indexing: A fast dimensionality reduction algorithm with applications to document retrieval and categorization. Technical Report TR-00-0016, University of Minnesota, 2000.

[83] Auguste Kerckhoffs. La cryptographie militaire. *Journal des sciences militaires*, 9:5–83(Jan.)& 161–191(Feb.), 1883.

[84] Reid Kerr and Robin Cohen. Smart cheaters do prosper: defeating trust and reputation systems. In *Proc. of the 8th Int. Conf. on Autonomous Agents & Multiagent Systems (AAMAS '09)*, pages 993–1000. IFAAMAS, 2009.

[85] Michael Kinateder, Ernesto Baschny, and Kurt Rothermel. Towards a generic trust model - comparison of various trust update algorithms. In *Proc. of the 3rd Int. Conf. on Trust Management (iTrust '05)*, pages 177–192. Springer-Verlag, 2005.

[86] Michael Kinateder and Kurt Rothermel. Architecture and algorithms for a distributed reputation system. In *Proc. of the 1st Int. Conf. on Trust Management (iTrust'03)*, pages 1–16. Springer-Verlag, 2003.

[87] Tomas B. Klos and Han La Poutré. Decentralized reputation-based trust for assessing agent reliability under aggregate feedback. In *Proc. of the 17th Belgium-Netherlands Conf. on Artificial Intelligence (BNAIC '05)*, pages 357–358, 2005.

[88] Derrick Kondo, Filipe Araujo, Paul Malecot, Patrício Domingues, Luís Moura Silva, Gilles Fedak, and Franck Cappello. Characterizing result errors in internet desktop grids. In *Euro-Par 2007 Parallel Processing: Proc. of the 13th European EURO-PAR Conference*, volume 4641 of *LNCS*, pages 361–371. Springer-Verlag, 2007.

[89] Justin Kruger and David Dunning. Unskilled and unaware of it: How difficulties in recognizing one's own incompetence lead to inflated self-assessments. *Journal of Personality and Social Psychology*, 77(6):1121–1134, 1999.

[90] Anukool Lakhina, Mark Crovella, and Christophe Diot. Diagnosing network-wide traffic anomalies. In *Proc. of the 2004 Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '04)*, pages 219–230. ACM, 2004.

[91] Anukool Lakhina, Mark Crovella, and Christophe Diot. Mining anomalies using traffic feature distributions. In *Proc. of the 2005 Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '05)*, pages 217–228. ACM, 2005.

[92] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The Byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

[93] Aleksandar Lazarevic, Levent Ertöz, Vipin Kumar, Aysel Ozgur, and Jaideep Srivastava. A comparative study of anomaly detection schemes in network intrusion detection. In *Proc. of the 3rd SIAM Int. Conf. on Data Mining (SDM '03)*, pages 25–36. SIAM, 2003.

[94] Vladimir Iosifovich Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics–Doklady*, 10(8):707–710, 1966.

[95] Brian Neil Levine, Clay Shields, and N. Boris Margolin. A survey of solutions to the Sybil attack. Technical Report 2006-052, University of Massachusetts Amherst, 2006.

[96] Jian Liang, Rakesh Kumar, Y. Xi, and Keith W. Ross. Pollution in P2P file sharing systems. In *Proc. of the 24th Annual Joint Conf. of the IEEE Computer and Communications Societies (INFOCOM '05)*, pages 1174–1185. IEEE, 2005.

[97] Churn-Jung Liau. Belief, information acquisition, and trust in multi-agent systems: a modal logic formulation. *Artif. Intell.*, 149(1):31–60, 2003.

[98] Seymour Lipschutz and Marc Lars Lipson. *Probability*. Schaum's Outline Series. McGraw-Hill, second edition, 2000.

[99] Stephen Paul Marsh. *Formalising Trust as a Computational Concept*. PhD thesis, University of Stirling, 1994.

[100] Sjouke Mauw and Martijn Oostdijk. Foundations of attack trees. In *Information Security and Cryptology: Revised Selected Papers of the 8th Int. ICISC Conference*, volume 3935 of *LNCS*, pages 186–198. Springer Verlag, 2006.

[101] D. Harrison McKnight and Norman L. Chervany. The meanings of trust. Technical report, University of Minnesota, 1996.

[102] Vaibhav Mehta, Constantinos Bartzis, Haifeng Zhu, Edmund M. Clarke, and Jeannette M. Wing. Ranking attack graphs. In *Recent Advances in Intrusion Detection: Proc. of the 9th Int. RAID Symposium*, volume 4219 of *LNCS*, pages 127–144. Springer-Verlag, 2006.

[103] Dejan S. Milojičić, Vana Kalogeraki, Rajan Lukose, Kiran Nagaraja, Jim Pruyne, Bruno Richard, Sami Rollins, and Zhichen Xu. Peer-to-peer computing. Technical Report HPL-2002-57 (R.1), HP Laboratories Palo Alto, 2002.

[104] Cleve B. Moler. A tale of two numbers. *SIAM News*, 28(1):16, January 1995.

[105] David Molnar. The SETI@home problem. *ACM Crossroads: E-commerce*, 7(1), 2000.

[106] Andrew P. Moore, Robert J. Ellison, and Richard C. Linger. Attack modeling for information security and survivability. Technical Report CMU/SEI-2001-TN-001, CMU Software Engineering Institute, 2001.

[107] David S. Moore. *The Basic Practice of Statistics*. W. H. Freeman & Co., New York, NY, USA, fourth edition, 2007.

[108] Greg Mori and Jitendra Malik. Recognizing objects in adversarial clutter: Breaking a visual CAPTCHA. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR '03)*, pages 134–144. IEEE Computer Society, 2003.

[109] Lik Mui, Mojdeh Mohtashemi, and Ari Halberstadt. A computational model of trust and reputation for e-businesses. In *Proc. of the 35th Annual Hawaii Int. Conf. on System Sciences (HICSS '02)*, pages 188–196. IEEE Computer Society, 2002.

[110] Guillaume Muller and Laurent Vercouter. L.I.A.R.: Achieving social control in open and decentralised multi-agent systems. Technical Report 2008-700-001, École des Mines de Saint-Étienne, 2008.

[111] Gia Hien Nguyen, Philippe Chatalic, and Marie-Christine Rousset. A probabilistic trust model for semantic peer-to-peer systems. In *Proc. of the 18th Eur. Conf. on Artificial Intelligence (ECAI '08)*, pages 881–882. IOS Press, 2008.

[112] Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V. Vazirani, editors. *Algorithmic Game Theory*. Cambridge University Press, 2007.

[113] Warren B. Powell. *Approximate Dynamic Programming*. John Wiley & Sons, Inc., first edition, 2007.

[114] Bartosz Przydatek, Dawn Song, and Adrian Perrig. SIA: secure information aggregation in sensor networks. In *Proc. of the 1st Int. Conf. on Embedded Networked Sensor Systems (SenSys '03)*, pages 255–265. ACM, 2003.

[115] W.V. Quine. A way to simplify truth functions. *American Mathematical Monthly*, 62(9):627–631, 1955.

[116] Raghu Ramakrishnan and Johannes Gehrke. *Database Management Systems*. McGraw-Hill, third edition, 2002.

[117] Sarvapali D. Ramchurn, T. D. Huynh, and Nicholas R. Jennings. Trust in multi-agent systems. *Knowl. Eng. Rev.*, 19(1):1–25, 2004.

[118] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. In *Proc. of the 2001 Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '01)*, pages 161–172. ACM, 2001.

[119] Martin Rehák and Michal Pěchouček. Trust modeling with context representation and generalized identities. In *Proc. of the 11th Int. Workshop on Cooperative Information Agents (CIA '07)*, pages 298–312. Springer-Verlag, 2007.

[120] Martin Rehák, Michal Pěchouček, Martin Grill, Jan Stiborek, Karel Bartoš, and Pavel Čeleda. Adaptive multiagent system for network traffic monitoring. *IEEE Intelligent Systems*, 24(3):16–25, 2009.

[121] Martin Rehák, Eugen Staab, Volker Fusenig, Michal Pěchouček, Martin Grill, Jan Stiborek, Karel Bartoš, and Thomas Engel. Runtime monitoring and dynamic reconfiguration for intrusion detection systems. In E. Kirda, S. Jha, and D. Balzarotti, editors, *Recent Advances in Intrusion Detection: Proc. of the 12th Int. Symposium On Recent Advances In Intrusion Detection (RAID '09)*, volume 5758 of *LNCS*, pages 61–80. Springer-Verlag, September 2009.

[122] Martin Rehák, Eugen Staab, Volker Fusenig, Jan Stiborek, Martin Grill, Karel Bartoš, Michal Pěchouček, and Thomas Engel. Threat-model driven runtime adaptation and evaluation of intrusion detection system. In *Proc. of the 6th Int. Conf. on Autonomic Computing and Communications (ICAC '09)*, pages 65–66. ACM, June 2009.

[123] Martin Rehák, Eugen Staab, Michal Pěchouček, Jan Stiborek, Martin Grill, and Karel Bartoš. Dynamic information source selection for intrusion detection systems. In K. S. Decker, J. S. Sichman, C. Sierra, and C. Castelfranchi, editors, *Proc. of the 8th Int. Conf. on Autonomous Agents & Multiagent Systems (AAMAS '09)*, pages 1009–1016. IFAAMAS, May 2009.

[124] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. GroupLens: an open architecture for collaborative filtering of netnews. In *Proc. of the 1994 ACM Conf. on Computer Supported Cooperative Work (CSCW '94)*, pages 175–186. ACM, 1994.

[125] Paul Resnick, Ko Kuwabara, Richard Zeckhauser, and Eric Friedman. Reputation systems. *Commun. ACM*, 43(12):45–48, 2000.

[126] C. J. Van Rijsbergen. *Information Retrieval*. Butterworth-Heinemann, Newton, MA, USA, 1979.

[127] Jean-Louis Roch and Sébastien Varrette. Probabilistic certification of divide & conquer algorithms on global computing platforms: application to fault-tolerant exact matrix-vector product. In *Proc. of the Int. Wksh. on Parallel Symbolic Computation (PASCO '07)*, pages 88–92. ACM, 2007.

[128] Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proc. of the IFIP/ACM Int. Conf. on Distributed Systems Platforms Heidelberg (Middleware '01)*, pages 329–350. Springer-Verlag, 2001.

[129] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, second edition, 2003.

[130] Jordi Sabater and Carles Sierra. REGRET: Reputation in gregarious societies. In *Proc. of the 4th Workshop on Deception, Fraud and Trust in Agent Societies*, pages 194–195. ACM Press, 2001.

[131] Jordi Sabater and Carles Sierra. Review on computational trust and reputation models. *Artif. Intell. Rev.*, 24(1):33–60, 2005.

[132] Luis F. G. Sarmenta. *Volunteer computing*. PhD thesis, Massachusetts Institute of Technology, 2001.

[133] Luis F. G. Sarmenta. Sabotage-tolerance mechanisms for volunteer computing systems. *Future Generation Comp. Syst.*, 18(4):561–572, 2002.

[134] Stefan Saroiu, P. Krishna Gummadi, and Steven D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proc. of the SPIE/ACM Conf. on Multimedia Computing and Networking (MMCN '02)*, 2002.

[135] Reginald E. Sawilla and Xinming Ou. Identifying critical attack assets in dependency attack graphs. In *Computer Security: Proc. of the 13th European ESORICS Symposium*, volume 5283 of *LNCS*, pages 18–34. Springer-Verlag, 2008.

[136] Nigel Shadbolt, Tim Berners-Lee, and Wendy Hall. The semantic web revisited. *IEEE Intelligent Systems*, 21(3):96–101, 2006.

[137] Oleg Sheyner, Joshua Haines, Somesh Jha, Richard Lippmann, and Jeannette M. Wing. Automated generation and analysis of attack graphs. In *Proc. of the 2002 IEEE Symposium on Security and Privacy (SP '02)*, page 273. IEEE Computer Society, 2002.

[138] Gheorghe Cosmin Silaghi, Filipe Araujo, Luis Moura Silva, Patricio Domingues, and Alvaro Arenas. Defeating colluding nodes in desktop grid computing platforms. In *Proc. of the 2nd Workshop on Desktop Grids and Volunteer Computing (PCGrid '08)*. IEEE Computer Society, 2008.

[139] Suresh Singh, Mike Woo, and C. S. Raghavendra. Power-aware routing in mobile ad hoc networks. In *Proc. of the 4th Annual ACM/IEEE Int. Conf. on Mobile Computing and Networking (MobiCom '98)*, pages 181–190. ACM, 1998.

[140] Michael Sipser. *Introduction to the Theory of Computation*. Int. Thomson Publishing, 1996.

[141] Murray R. Spiegel, John Schiller, and R. Alu Srinivasan. *Probability and Statistics*. Schaum's Outline Series. McGraw-Hill, third edition, 2009.

[142] Avinash Sridharan, Tao Ye, and Supratik Bhattacharyya. Connectionless port scan detection on the backbone. In *Proc. of the 25th IEEE Int. Performance Computing and Communications Conference (IPCCC '06), Workshop on Malware*, pages 567–576. IEEE, 2006.

[143] Eugen Staab and Martin Caminada. On the profitability of incompetence. In *Proc. of the 11th Workshop on Multi-Agent-Based Simulation (MABS '10)*, 2010.

[144] Eugen Staab and Thomas Engel. Combining cognitive and computational concepts for experience-based trust reasoning. In Rino Falcone, Suzanne Barber, Jordi Sabater, and Munindar Singh, editors, *Proc. of the 11th Int. Workshop on Trust in Agent Societies (TRUST '08)*, pages 41–45. IFAAMAS, May 2008.

[145] Eugen Staab and Thomas Engel. Combining cognitive with computational trust reasoning. In Rino Falcone, Suzanne Barber, Jordi Sabater, and Munindar Singh, editors, *Trust in Agent Societies: Revised Selected and Invited Papers of the 11th Int. Workshop on Trust in Agent Societies (TRUST '08)*, volume 5396 of *LNCS (LNAI)*, pages 99–111. Springer-Verlag, December 2008.

[146] Eugen Staab and Thomas Engel. Collusion detection for distributed computing. In Sjouke Mauw and Leendert van der Torre, editors, *Proc. of the 1st Luxembourg Day on Security and Reliability (P1Day '09)*, page 75. University of Luxembourg, February 2009.

[147] Eugen Staab and Thomas Engel. Collusion detection for grid computing. In Franck Cappello, Cho-Li Wang, and Rajkumar Buyya, editors, *Proc. of the 9th IEEE/ACM Int. Symposium on Cluster Computing and the Grid (CCGrid '09)*, pages 412–419. IEEE Computer Society, May 2009.

[148] Eugen Staab and Thomas Engel. Formalizing excusableness of failures in multi-agent systems. In Aditya Ghose, Guido Governatori, and Ramakoti Sadananda, editors, *Agent Computing and Multi-Agent Systems: Revised Papers of the 10th Pacific Rim Int. Conf. on Multi-Agents (PRIMA '07)*, volume 5044 of *LNCS (LNAI)*, pages 122–133. Springer-Verlag, November 2009.

[149] Eugen Staab and Thomas Engel. Tuning evidence-based trust models. In *Proc. of the 2009 IEEE Int. Conf. on Information Privacy, Security, Risk and Trust (PASSAT '09)*, pages 92–99. IEEE Computer Society, August 2009.

[150] Eugen Staab, Volker Fusenig, and Thomas Engel. Towards trust-based acquisition of unverifiable information. In Matthias Klusch, Michal Pěchouček, and Axel Polleres, editors, *Cooperative Information Agents XII: Proc. of the 12th Int. Workshop on Cooperative Information Agents (CIA '08)*, volume 5180 of *LNCS (LNAI)*, pages 41–54. Springer-Verlag, September 2008.

[151] Eugen Staab, Volker Fusenig, and Thomas Engel. Trust-aided acquisition of unverifiable information. In Malik Ghallab, Constantine D. Spyropoulos, Nikos Fakotakis, and Nikolaos M. Avouris, editors, *Proc. of the 18th European Conf. on Artificial Intelligence (ECAI '08)*, volume 178 of *Frontiers in Artificial Intelligence and Applications*, pages 869–870. IOS Press, July 2008.

[152] Eugen Staab, Volker Fusenig, and Thomas Engel. Using correlation for collusion detection in grid settings. Technical Report 000657499, University of Luxembourg, July 2008.

[153] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. of the 2001 Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '01)*, pages 149–160. ACM, 2001.

[154] Daniel Stutzbach and Reza Rejaie. Understanding churn in peer-to-peer networks. In *Proc. of the 6th ACM SIGCOMM Conf. on Internet Measurement (IMC '06)*, pages 189–202. ACM, 2006.

[155] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[156] M. Taufer, D. Anderson, P. Cicotti, and C. L. Brooks III. Homogeneous redundancy: a technique to ensure integrity of molecular simulation results using public computing. In *Proc. of the 19th IEEE Int. Parallel and Distributed Processing Symposium (IPDPS '05) - Workshop 1*, page 119.1. IEEE Computer Society, 2005.

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

[158] C.-K. Toh. *Ad Hoc Wireless Networks: Protocols and Systems*. Prentice Hall PTR, 2001.

[159] Stijn van Dongen. *Graph Clustering by Flow Simulation*. PhD thesis, University of Utrecht, 2000.

[160] Pedro B. Velloso, Rafael P. Laufer, Otto Carlos M. B. Duarte, and Guy Pujolle. A trust model robust to slander attacks in ad hoc networks. In *Proc. of the Workshop in Advanced Networking and Communications (ANC), at ICCCN '08*, 2008.

[161] Mohit Virendra, Murtuza Jadliwala, Madhusudhanan Chandrasekaran, and Shambhu Upadhyaya. Quantifying trust in mobile ad-hoc networks. In *Proc. of the IEEE Int. Conf. on Integration of Knowledge Intensive Multiagent Systems (KIMAS '05)*, pages 65–70. IEEE Computer Society, 2005.

[162] Luis von Ahn, Manuel Blum, Nicholas J. Hopper, and John Langford. CAPTCHA: Using hard AI problems for security. In *Advances in Cryptology: Proc. of the Int. EUROCRYPT Conference*, volume 2656 of *LNCS*, pages 294–311. Springer-Verlag, 2003.

[163] Luis von Ahn, Benjamin Maurer, Colin McMillen, David Abraham, and Manuel Blum. reCAPTCHA: human-based character recognition via web security measures. *Science*, 321(5895):1465–1468, 2008.

[164] Ulrike von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17:395–416, 2007.

[165] David Wagner. Resilient aggregation in sensor networks. In *Proc. of the 2nd ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN '04)*, pages 78–87. ACM, 2004.

[166] Yao Wang and Julita Vassileva. Trust and reputation model in peer-to-peer networks. In *Proc. of the 3rd Int. Conf. on Peer-to-Peer Computing (P2P '03)*, page 150. IEEE Computer Society, 2003.

[167] Yonghong Wang and Munindar P. Singh. Formal trust model for multiagent systems. In *Proc. of the 20th Int. Joint Conf. on Artificial Intelligence (IJCAI '07)*, pages 1551–1556, 2007.

[168] Hal Wasserman and Manuel Blum. Software reliability via run-time result-checking. *J. ACM*, 44(6):826–849, 1997.

[169] Gerhard Weiss, editor. *Multiagent systems: a modern approach to distributed artificial intelligence*. MIT Press, Cambridge, MA, USA, 1999.

[170] Eric W. Weisstein. Beta distribution. `http://mathworld.wolfram.com/BetaDistribution.html`. [From MathWorld–A Wolfram Web Resource; accessed 2010-01-25].

[171] Eric W. Weisstein. Beta function. `http://mathworld.wolfram.com/BetaFunction.html`. [From MathWorld–A Wolfram Web Resource; accessed 2010-01-25].

[172] Andrew Whitby, Audun Jøsang, and Jadwiga Indulska. Filtering out unfair ratings in Bayesian reputation systems. In *Proc. of the 7th Int. Workshop on Trust in Agent Societies (at AAMAS '04)*, 2004.

[173] Jonathan Wilkins. Strong CAPTCHA guidelines, v1.2. `http://www.bitland.net/captcha.pdf`, December 2009. [Online; accessed 2010-02-22].

[174] Mark Witkowski and Jeremy Pitt. Objective trust-based agents: Trust and trustworthiness in a multi-agent trading society. In *Proc. of the 4th Int. Conf. on Multi-Agent Systems (ICMAS '00)*, pages 463–464. IEEE Computer Society, 2000.

[175] Michael Woolridge. *Introduction to Multiagent Systems*. John Wiley & Sons, Inc., 2001.

[176] Li Xiong and Ling Liu. A reputation-based trust model for peer-to-peer ecommerce communities. In *Proc. of the 5th Int. Conf. on Electronic Commerce (EC '03)*, pages 228–229. ACM, 2003.

[177] Li Xiong and Ling Liu. Peertrust: Supporting reputation-based trust for peer-to-peer electronic communities. *IEEE Trans. Knowl. Data Eng.*, 16(7):843–857, 2004.

[178] Kuai Xu, Zhi-Li Zhang, and Supratik Bhattacharyya. Reducing unwanted traffic in a backbone network. In *Proc. of the Steps to Reducing Unwanted Traffic on the Internet Workshop (SRUTI '05)*, pages 8–14. USENIX Association, 2005.

[179] Jeff Yan and Ahmad Salah El Ahmad. A low-cost attack on a Microsoft CAPTCHA. In *Proc. of the 15th ACM Conf. on Computer and Communications Security (CCS '08)*, pages 543–554. ACM, 2008.

[180] Bin Yu and Munindar P. Singh. Detecting deception in reputation management. In *Proc. of the 2nd Int. Joint Conf. on Autonomous Agents & Multiagent Systems (AAMAS '03)*, pages 73–80. ACM, 2003.

[181] Matthew Yurkewych, Brian N. Levine, and Arnold L. Rosenberg. On the cost-ineffectiveness of redundancy in commercial P2P computing. In *Proc. of the 12th Conf. on Computer and Communications Security (CCS '05)*, pages 280–288. ACM, 2005.

[182] Giorgos Zacharia. Collaborative reputation mechanisms for online communities. Master's thesis, Massachusetts Institute of Technology, 1999.

[183] Min Zhang, Wolfgang John, K. Claffy, and Nevil Brownlee. State of the art in traffic classification: A research review. In *Proc. of the 10th Int. Conf. on Passive and Active Measurement (PAM '09), Student Workshop*, 2009.

[184] Ben Y. Zhao, John D. Kubiatowicz, and Anthony D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, University of California at Berkeley, 2001.

[185] Shanyu Zhao, Virginia Lo, and Chris GauthierDickey. Result verification and trust-based scheduling in peer-to-peer grids. In *Proc. of the 5th IEEE Int. Conf. on Peer-to-Peer Computing (P2P '05)*, pages 31–38. IEEE Computer Society, 2005.

[186] Charikleia Zouridaki, Brian L. Mark, Marek Hejmo, and Roshan K. Thomas. Robust cooperative trust establishment for MANETs. In *Proc. of the 4th ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN '06)*, pages 23–34. ACM, 2006.