TECHNISCHE UNIVERSITÄT
CHEMNITZ

Department of Computer Science
Distributed and Self-organizing Systems Group

Bachelor's Thesis

Content Trust Evaluation for Web Applications

Arved Kirchhoff

Chemnitz, 4 August 2022

Examiner: Prof. Dr.-Ing. Martin Gaedke
Supervisor: Valentin Siegert

Kirchhoff, Arved
Content Trust Evaluation for Web Applications
Bachelor's Thesis, Department of Computer Science
Technische Universität Chemnitz, August 2022

Acknowledgements

I want to express my gratitude towards everyone who supported me in the process of writing my bachelor's thesis. In particular, I would like to thank the following people:

Valentin Siegert

Vivien Rehrl

Arndt Kirchhoff

Abstract

Based around the advancement of privacy protection and helping individuals to regain control over their personal data, there have been efforts and initiatives towards the redecentralization of the web. To deal with the large amount of data providers in decentralized systems, it will be necessary to use web applications that automatically select a sufficiently trustworthy source of information for the given task. This requires those web applications to make trust-aware decisions, which, in turn, requires a framework to adequately evaluate and judge trust automatically. An already existing approach for this trust judgement is called entity trust. Entity trust works by combining data about the reputation of an information source with authentication technologies. If the information source is being trusted, the information will be trusted as well. Consequentially, the result of a trust evaluation using entity trust is absolute and a comparison of trust between two sources of information is not possible. Furthermore, this form of trust judgement does not consider the context or content of the evaluated information. Therefore, content trust is possibly a better approach that considers data about the content and context of an evaluated resource to create a basis for more nuanced trust decisions. Content trust realizes this by evaluating 19 influencing factors to calculate a final trust value. The objective of this thesis is to develop and implement a framework that allows web applications to independently evaluate the trustworthiness of information by using a content trust centered approach. Instead of a simple dichotomous answer to the question whether the information should be trusted or distrusted, the framework will produce a value that enables a more complex and nuanced trust-aware decision by the web application. This thesis solely focuses on a first implementation prototype for the trust evaluation of the 19 factors of content trust. The trust scale used by the framework should be exchangeable and tested with the scale from Marsh and Briggs. Further, the framework design requires to evaluate all 19 different factors based on specific input information per factor. These can be given by expert knowledge and described for evaluations with the aTLAS scenarios, as the input information creation at runtime is not part of this thesis. The characteristics and details of trust-aware decisions will not be discussed either. The objective of this thesis is to find an approach or combination of approaches for the previously mentioned problems and tasks in the context of content trust for web applications in a redecentralized web. This particularly includes the state of the art regarding trust frameworks for web applications. The demonstration of feasibility with an implementation prototype of the concept is part of this thesis as well as a suitable evaluation with exemplary use cases.

Contents

List of Figures

List of Tables

List of Listings

1 Introduction

Initially, the web was envisioned and developed with decentralization and universality in mind, meaning a web without central administrative structures, and no authorities controlling or limiting the access to the web for anyone. To ensure that the web could be a means of communication for anyone, independent of social, cultural, or political backgrounds, there was supposed to be a universal language computers and applications "speak" to each other to establish universal compatibility. However, beginning in the late 1990s the web grew into an unparalleled economic opportunity, reshaping it drastically. [1]

1.1 Current Situation

There are strong economic incentives for companies operating within the web to create their very own centralized ecosystem. For example, centralized solutions tend to be more efficient in processing data, both for providing services to users, as well as collecting valuable user data [1].

As Ibáñez et al. [1] describe, having an universal language for communication between computers is great for ensuring compatibility between devices. However, it is more profitable for service providers to force their users into a company owned and operated ecosystem with proprietary data formats and application interfaces. Keeping their users within this system provides the company with a market advantage over their competitors by hindering the transfer of data and therefore incentivizing users to continue using their services to work with their existing data [2].

Because the data formats and application interfaces are controlled by a single entity, this entity has the authority to change them at its discretion and can therefore possibly render applications that enable easier data transfer obsolete [2]. The effort required by users for transferring their data to another service provider leads to a situation where new competitors can barely enter the market. Hence, promoting further centralization and creation of service monopolies, giving the few companies more control over the user‘s data and the way users interact with data that belongs to them. The essentially nonexistent control users have over their own data is the reason for strong privacy concerns [2, 1] and led to the formation of initiatives for

re-decentralizing the web, e.g. SoLiD [2], which is a project led by Tim Berners-Lee, and the Next Generation Internet initiative operated by the EU[1].

Web applications often need to rely on external data and services to provide their own functionality. However, because interaction partners may stand to gain an advantage from exploiting or abusing this interaction, it is crucial to introduce a concept comparable to human trust to those interactions. Trust allows web applications to carefully choose their interaction partner and prevent abuse of their functionality. [3] In a centralized system, trust relationships between potential interaction partners are controlled and created by a central authority that both parties trust [4]. This authority considers a third party trustworthy (or not) based on the validation of predetermined artifacts or manual, human-given permission [4]. Authentication mechanisms are used to ensure the identity of the third-parties during the interaction [5].

1.2 Motivation

A solution for strengthening privacy is giving individuals more control over how their data is accessed and used by services and applications. This could be done by re-decentralizing the web with universal, open protocols and application interfaces, and allowing the user to be independent of central authorities or entities. [1, 2, 4] Users would store their data on their own devices or, for example, in a decentralized pod-like structure as proposed by SoLiD [2], and grant applications access to their data based on a fine-grained, universal access control system, allowing the user to prevent any application from accessing their data at any given point. The advantage of storing the data in an universal data format is that, should a user not be satisfied with an application for any reason, they can simply use any other application without having to migrate their existing data from one service to another [2].

Storing and handling personal data like this enforces transparency over which data an applications is using. Further, it enhances user privacy by giving every user the necessary information for making an informed decision on their usage of the application. Furthermore, it also provides a user with more freedom of action by making the ability to switch applications easy by design [4].

As described by Siegert et al. [4], a re-decentralized web will have more data providers, which can be added, removed or changed at any time. Some data providers may be hidden, or indirectly accessed, and some may have malicious or exploitative intentions. Yet, web applications should still be able to interact with an arbitrar-

[1] https://digital-strategy.ec.europa.eu/en/policies/next-generation-internet-initiative

ily large amount of interaction partners without having to compromise on security or functionality. To achieve this, those web applications need an understanding of trust and a decentralized trust management system that uses a trust model which evaluates trust more dynamically than direct or reputation-based trust models [5].

Trust models based on direct trust use directly observed or experienced historical interaction outcomes as evidence for evaluating the trustworthiness of potential interaction partners [3, 6]. However, this means that trust testimonials made by other participants in the network are not taken into account. In contrast, reputation-based trust describes a form of trust management based on trust testimonials [4, 6]. To understand how modern reputation-based trust works in a centralized web, it is helpful to look at the example of Transport Layer Security (TLS) as specified in [7]. The main goal of TLS is to encrypt web traffic, but the user needs a way of ensuring that the received public key really came from the desired web application and not a malicious actor. This is done via a signature chain that leads up to a root certificate authority (CA). If this root CA is trusted, every TLS certificate that has a signature chain leading up to this CA is trusted to be valid for the web resource it belongs to. This is reputation-trust because the trust testimonials come from the root certificate authorities a user trusts. Nowadays, this validation is done automatically and transparently by browsers. However, if a requested web resource presents a TLS certificate that is not signed by a known root CA, or the signature chain described in the certificate is broken, the automatic validation will fail and the decision whether to trust the certificate is outsourced to the human user. This works for a centralized web where web resources were chosen manually by the user. But, in a decentralized web with a plethora of information providers and web applications that are intended to work autonomously, it would be more expensive and potentially slower to employ humans than equipping the web application to compensate for missing trust testimonials itself [4].

1.3 Problem Analysis

Creating trust relationships with the help of a central authority means that the interaction partners have a trust dependency on those centralized authorities. However, those authorities do not exist in a decentralized web. Furthermore, an important characteristic of a decentralized web is that, because everyone can share information without needing permission, there is a multitude of uncontrolled data providers available, whose information quality may be questionable or even tainted by malicious intent, e.g. fake news or propaganda [5, 4]. Therefore it is essential to filter provided information based on their trustworthiness. This cannot be done manually given the large and potentially uncontrollably growing amount of data providers, as well as

the fact that those providers might be hidden or not exposed to the user. Hence, there needs to be an automated solution for determining whether a given piece of information should be trusted or not, ideally without human-in-the-loop. [4]

Deciding whether a given piece of information should be trusted consists of two steps [3, 4]:

1. **Trust Evaluation** means the assessment of the trustworthiness of a potential interaction partner or resource. Because an interaction partner may appear, disappear or behave differently at any given time, this assessment should consider the current situation dynamically, and preferably determine the trustworthiness for each piece of information within its context. At the end of the evaluation process, a trust value is created, which rates the assessed trustworthiness based on the used trust scale. The format of this trust scale is arbitrary and can be binary, with the only possible values being "trusted" or "distrusted", or more nuanced by utilizing a value on a numerical spectrum.

2. **Trust-Aware Decision** describes a decision based on the trust values resulting from the trust evaluation. Depending on the context and use case of the web application, this may mean simply selecting the most trustworthy potential interaction partner. In other cases, potential interaction partners may be compared and ranked according to their trustworthiness, or the interaction may be canceled entirely if a certain trust threshold is not met. A trust scale that implements this kind of threshold was described by Marsh and Briggs in [8]. Their trust scale uses a numerical trust value between -1 and 1, where a negative trust value equals distrust and a positive value equals trust. Yet, it is not implied that the evaluating web application automatically trusts a potential interaction partner if their trust value is positive. This is because there are situations where an application trusts another application to some degree, but not enough to complete the current interaction – e.g., you may trust your neighbor to water your plants, but would not allow them to handle your financial transactions. In this case, the trust value of the other application would be below the *cooperation threshold*.

An already available and well researched group of models for trust evaluation is *entity trust* [4, 5]. Entity trust based trust evaluation works by combining reputation and authentication where reputation describes the identity and behavior of an entity, based on evidence derived from recent interactions. This approach suffices for creating trust relationships within centralized systems. However, decentralized networks are more dynamic and less transparent than centralized ones. Performing data aggregation within a decentralized environment requires relying on an arbitrarily large number of information providers where the original data sources might be hidden or unknown. Furthermore, any network participant may provide high quality informa-

tion, but there is no structural authority that provides trust factors for evaluating the trustworthiness of a piece of information. Therefore, the data‘s potential harmfulness should be considered. [5, 4]

As written by both Siegert et al. [4] and Gil et al. [5], this dynamic characteristic of a decentralized network is not adequately dealt with in the scope of entity trust-based trust management approaches. That is because contextual and contentual properties of a certain piece of information are not considered. This results in blanket statements about the trustworthiness of an entity, which means trusting the entity and any data it sends. This is in contrast to evaluating and expressing the trust towards an entity within the context of a certain piece of information as well as the data aggregation context. The latter is referred to as *content trust* [5]. However, even if entity trust does not suffice as sole trust evaluation approach, it is still a solid base for trust evaluation. Therefore, content trust builds on entity trust and combines it with contextual and contentual considerations.

Content trust is subjective in a way that its factors for trust evaluation are less clear than with entity trust and might differ depending on the regarded use case. This thesis defines the extent of content trust based on the 19 factors described by Gil et al. [5]. In theory those content trust factors are able to fulfill the requirements of a situational aware trust evaluation. However, it is not yet clear which content trust factors are relevant within the context of web applications and how they can be implemented for this use case.

1.4 Objectives and Limitations

The main objective of this thesis is to provide a way for web applications to use content trust-based, automated, and decentralized trust evaluation, similar to the description of content trust by Gil et al. in [5]. The trust evaluation should be able to work without the need for human interaction. As outlined in chapter 1.3, it is not yet solved how each of the 19 factors of content trust named by the authors can be implemented within the context of web applications. Therefore, the first step is to analyze each factor to decide to what extent it can be implemented for web applications. Based on the results of this analysis, a theoretical model for trust evaluation will be created, which will then be implemented within a trust evaluation framework. As explained by Siegert et al. [4], the decentralized web can be considered as an open and dynamic multi-agent system. For this reason, research results from this area can be well incorporated into the following considerations.

The focus of this thesis' attention is developing the described trust framework. For this reason, the classification and sanitization of the required input data is not dis-

cussed within this thesis and will be left as subject of future work. For example, the content trust factor *Age*, which describes the time of creation for the evaluated piece of information [5], obviously requires some kind of input data detailing when this piece of information was created. The portrayed framework expects this data to be parsed, understood, and formatted a certain way to work as expected, e.g., a timestamp. However, the requested web resource may provide its date of creation in a format based on ISO 8601. The code that is running the framework has to handle the proper format conversion et cetera.

The same goes for the utilization of the trust value that is created through the trust evaluation. This thesis will not discuss details regarding the trust-aware decision or the usage of the resulting trust value in general – the focus of this thesis lies solely on trust evaluation.

2 State of the Art

The purpose of this chapter is to systemize and analyze the current state of the art in trust evaluation. This is done by first presenting a catalog of requirements, as well as the corresponding rating scales, on the basis of which the fulfillment of the requirements is expressed. Afterwards, a categorization for trust evaluation models will be introduced and representative trust models for each category will be briefly showcased. Finally, the presented categories will be compared based on the requirements stated in chapter 2.1 and drawn conclusions will be described.

2.1 Identified Requirements

The following set of requirements is derived from the problem described in chapter 1.3 with regard to the desired outcome as described in chapter 1.2. These requirements aim at determining the best model for trust evaluation.

2.1.1 R1 - Trust Evaluation time

The evaluation of trust is an extensive and challenging problem. For this reason, trust models often present complex solutions with many detailed considerations. However, within the decentralized web an interaction between two web applications will only occur, if both parties deem the other to be trustworthy. This means that for every planned interaction, an application will have to evaluate the trust for an arbitrary number of potential interaction partners. This takes time and consumes computational resources that cannot be applied for solving the problem the application is used for, which is clearly undesirable.

To keep this kind of overhead low, the time that a trust model needs per trust evaluation should be as short as possible. But, runtime is depending on the specific use case, hardware and environmental conditions, which means it does not represent an appropriate metric for comparison in this context. Therefore, this requirement describes the necessary time complexity per trust evaluation. The complexity is expressed in big O notation where n describes the number of web applications within the network. The necessary data for this requirement is derived from statements

about the time complexity made by the respective authors, manual analysis of given pseudo code, or estimated on the basis of the described sequence of events.

This requirement will not be rated on a scale, but instead presented as is, or omitted if the complexity could not be determined or estimated.

2.1.2 R2 - Decentralization

Using a trust model with a centralized structure for evaluating trust in a decentralized web obviously defeats the purpose of decentralization and should therefore be avoided. However, centralization is, within the scope of this thesis, not considered absolute and will be split into two forms: 1) there is a small number of central authorities or infrastructure that control or influence the evaluation of trust or 2) there is no central authority or infrastructures, but the power distribution between web applications is unequal, in the sense that some applications have a larger system-given influence on a trust evaluation in the network than others. While both forms of centralization are undesirable, the first one can have a considerably more devastating negative impact if it were to be corrupted, and must therefore be rated lower than the second form.

The resulting scale, by which the decentralization of trust models is to be evaluated, is given in table 2.1.

Rating	Description
★★★	The trust evaluation is either completely independent or relies on the help (e.g reviews, witness testimony) from other, equally non-privileged web applications.
★★☆	There is an unequal systemic power distribution, where some web applications (e.g. group leaders, controllers) exert more influence over trust evaluations within the network than others. However, those more powerful applications may only exert their influence over small aspects of each trust evaluation or groups of subordinated web applications.
★☆☆	The model relies on few central authorities or pieces of infrastructure that control large parts, if not all of the trust evaluation.
-	The requirement does not apply or could not be determined.

Table 2.1: Rating scale for the requirement R2

2.1.3 R3 - Need For Human Action

As described in the introductory chapter, it is not feasible for service providers in a decentralized web to employ humans to evaluate trust manually because of the massive amount of potential interaction partners [4]. Similarly to the requirement R1, which aims at keeping the temporal overhead for trust evaluations low, the intention behind this requirement is to lower the amount of financial overhead generated by requiring humans to be involved in the trust evaluation.

Human action in this context means any work done by humans within the scope of the trust model. This excludes work like installing the trust model for the web application to use. Furthermore, human action that is based on the presented use case of a trust model will also be excluded from this requirement to allow for a fair comparison. For example, when using a reputation based trust model for an online marketplace, testimony can only be given after a human examined the product manually. However, if the same model was to be applied in an internet of things (IoT) environment, depending on the type of interaction, testimony could be created automatically. The human action required for creating testimony in the former use case would not be considered during the analysis.

The need for human action will be evaluated based on information gathered through manually analyzing the compared trust models. The results of the analysis will be rated by means of the rating scale given in table 2.2.

Rating	Description
★★★★	Human action is not required by the trust model.
★★★☆	Human action is solely required for the setup or initial phase of the trust model (e.g training a machine learning model, defining rules).
★★☆☆	Human action is required regularly (e.g. as fallback in case the automatic trust evaluation fails), but not for every interaction.
★☆☆☆	Human action is required for every interaction or trust evaluation.
-	The requirement does not apply or could not be determined.

Table 2.2: Rating scale for the requirement R3

2.1.4 R4 - Situational Awareness

Trust in the decentralized web is dynamic [4] and therefore highly situational. This means, that to properly assess trust, a trust model needs to analyze the situation in which the planned interaction is supposed to happen. Thus, contextual and contentual information should be considered within the trust evaluation process of the given trust model. In the following chapters, these considerations are summarized as *situational awareness*.

To categorize and rate the situational awareness shown by the compared trust models, the rating scale shown in table 2.3 will be applied. The necessary data for this will be provided by analyzing the models manually.

Rating	Description
★★★★	Both, the context as well as the content of an interaction are considered for the trust evaluation.
★★★☆	The context of an interaction is analyzed extensively within the trust evaluation or in an advanced way (e.g. behavior analysis, meta-analysis of ratings given by the potential interaction partner).
★★☆☆	The context of an interaction is analyzed trivially within the trust evaluation (e.g. measuring the response time of the potential interaction partner).
★☆☆☆	Neither the context nor the content of an interaction are considered at all.
-	The requirement does not apply or could not be determined.

Table 2.3: Rating scale for the requirement R4

2.1.5 R5 - Trust Scale

As stated by Marsh and Briggs [8], trust is not necessarily absolute or unconditional. Therefore, a well-crafted trust scale should empower the trust evaluation to create trust values that capture some of the dynamics of trust and, ideally, allow for an arbitrarily precise trust value that makes comparing potential interaction partners easier and more productive. This does not work with binary trust scales that can only differentiate between the states *trusted* and *not trusted*. A greater set of trust states that also incorporates states of high and low trust, ignorance or dis- and mistrust

[8] helps with making better decisions, but may not work for some use cases. This makes a meticulous trust scale a valid requirement for trust models to have.

The rating scale for evaluating the trust scales is given in table 2.4 and the information that poses the basis for this rating is acquired through an analysis of the compared models.

Rating	Description
★★★	The trust scale uses numerical values that allow for an arbitrarily high precision of the trust value (e.g. a value in $[0, 1]$).
★★☆	The trust scale uses a set of different trust states that will be assigned to the trust value. The set size is greater than 2 (e.g. an enum with the possible values of "highly trusted", "slightly trusted", "distrusted", and "missing information").
★☆☆	The trust scale is binary and only allows for expressing trust or no trust.
-	The requirement does not apply or could not be determined.

Table 2.4: Rating scale for the requirement R5

2.2 Systematization and Analysis

This chapter will present a grouping scheme for trust evaluation models that is based on the systematization concept for trust management systems introduced by Yu et al. [3]. But, while their concept exists with six different groups, this thesis only considers four of those groups to match the scope of this thesis, that is only focused on the process of trust evaluation and not the entire trust-aware decision, as laid out in the introductory chapter. The introduced groups classify different approaches to trust evaluation loosely based on the source of information a trust model primarily uses. Information sources for trust models are used to extract data from the network that can later be used to calculate trust values [6].

In their appearing order, those groups are:

1. Direct Trust-based trust models
2. Reputation-based trust models
3. Socio-Cognitive trust models

4. Organizational trust models

It should be noted however, that organizational trust models are not classified as such because of their information source, but because of their approach of delegating their trust decision towards an organizational structure, as explained in chapter 2.2.4. Furthermore, because of the characteristics of the information sources by themselves, it is often possible to achieve better and more accurate trust evaluation results if multiple information sources are used in combination [3, 6, 9]. Especially reputation-based trust models need underlying direct experience mechanisms because, without a record of the interaction data, no witness testimony could be shared. For this reason, there exist some overlaps – of models that use multiple information sources – between the groups. In those cases, the model has been sorted into the group of their respective primary or most relevant information source.

2.2.1 Direct Trust-based Trust Models

Direct trust-based trust models aggregate and analyze historical interaction outcomes to collect evidence for the trustworthiness of a potential interaction partner [3]. This evidence is also referred to as *direct experience* [6]. Direct experience is currently considered the most reliable and relevant information source for trust evaluation [9], and is a concept that is deeply rooted in the characteristics of human trust [3] – e.g., if a person made bad experiences trusting an interaction partner in the past, they will be less likely to trust the same interaction partner in the future.

Direct experience can be split into two groups of evidence: **direct interaction** and **direct observation** [6].

Direct interaction describes the situation that two participants of a network have directly interacted with each other. For example, web application **A** required a service or a certain piece of information from application **B**. The interaction occurred and **A** (as well as **B**) will remember the outcome of the interaction from now on and use this information in trust evaluations for future potential interactions. [6] This way of acquiring evidence is fairly straightforward and robust [9], but is not used as the sole source of information by many trust models [6]. That is, because creating trust through direct interaction requires numerous past interactions with the same web application. This is not viable in networks with a high number of participants – like the decentralized web – due to performance reasons and the overabundance of available potential interaction partners. Combined with the open and dynamic properties of the decentralized web, it is clear that direct interaction is not sufficient on its own. Hence, most trust models that use direct interactions combine it with other, more dynamic and easily available sources of information. [6]

In contrast, direct observation works by observing the interactions of others. That means, that applications **A** and **B** did not necessarily interact with each other before, but application **A** has observed **B**'s behavior in recent or current interactions with other participants of the network and remembers these outcomes. Observing an interaction is less performance intensive for **A** than interacting with **B** directly [3], but direct observation shows similar problems as direct interaction in terms of the number of participants in the network and the need for an extensive interaction history within the network to gather sufficient evidence for a balanced trust evaluation. Furthermore, the use of direct observation requires that the network it is used in has appropriate security measures put in place to prevent exposing interaction data or the identities of the interacting applications. [6] Nowadays, direct observation is often used in low-performance environments – e.g., IoT or autonomous vehicles – where network nodes make simple, often contextual observations – e.g., a potential interaction partners packet loss or packet error rates – to collect evidence and draw conclusions from those observations. Exemplary trust models that exhibit this kind of mechanism are described by Zhang et al. [10] and Arifeen et al. [11].

Analysis To determine this group's capability to fulfill the posed requirements, two direct trust-based trust models have been analyzed. The results of the analysis of both models based on the requirements described in chapter 2.1 are shown in table 2.5. Both approaches assume that trust can be considered a multi-objective optimization problem.

SD-TDQL is a trust management system for vehicular ad hoc networks and was introduced by Zhang et al. [10] in 2020. The trust evaluation model that is part of this architecture utilizes the count of forwarded data packets compared to the number of sent data packets to determine the link quality within the network. It is assumed that a compromised or untrustworthy device will not properly forward sent packets and therefore impact the link quality which can thus be used as indicator for untrustworthy network nodes.

The second trust model as presented by Jiang et al. [12] in 2012 uses an evolutionary optimization algorithm to decide which potential interaction partner should be interacted with. The underlying trust evaluation that calculates the necessary data for this decision is based on a Beta probability density function that is parameterized through direct interaction evidence, which, in this case, is the amount of successful and unsuccessful interactions in the past. The resulting trust value for a web application is the Beta distribution‘s probability expectation value, which describes the relative probability for successful future interaction outcomes.

Trust Model	R1	R2	R3	R4	R5
Zhang et al. [10]	$O(n)$	★★★	★★★★	★☆☆☆	★★★
Jiang et al. [12]	-	★★★	★★★☆	★☆☆☆	★★★
Total	$O(n)$	★★★	★★★☆ to ★★★★	★☆☆☆	★★★

Table 2.5: Analysis results for Direct Trust-based trust models

2.2.2 Reputation-based Trust Models

Reputation-based trust, also referred to as indirect experience or witness information, describes the aggregation of information about the trustworthiness of an application based on recommendations, opinions and witness testimony given by third-party applications [6, 13]. As explained in chapter 2.2.1, direct experience may not always be available in an abundance that allows confident trust-aware decisions [3], therefore a lot of trust models fill the gap by using witness testimony [3, 6, 9]. Existing implementations of the reputation models differ and, depending on the model specifics, witnesses must have had direct experiences with the target web application [3] or may relay testimony from other witnesses [9]. Witness testimony exist in two different types of information: (1) plain records of an interaction, similar to the ones the application would create itself while collecting direct experiences, and (2) recommendations [6].

However, witness information is less reliable than direct experience [3, 6, 9] and there are a number of different attack scenarios that exploit the reliance on other participants of the network [14]. This can affect trust-aware decisions negatively and requires trust models to implement systems that mitigate the negative effects of *Self-Promotion*, *Ballot-Stuffing*, *Bad-Mouthing* and similar, trust-based attacks [3, 14]. To combat this problem, reputation-based trust models mostly perform their trust evaluations in two steps, with the first step being the **trust evidence aggregation**, followed by the **testimony filtering** [3].

During the trust evidence aggregation, the evaluating web applications attempt to collect as much information and evidence about the potential interaction partner as possible. Because of the low reliability of indirect experiences, a majority of trust models weigh the collected witness testimonies against direct experiences and calculate an average. Models differ in their implementation of the weight, with some using a static weight value that remains constant over time, while others adapt the value according to the collected evidence. [3] This is done, because in a system without biased testimony, third-party evidence and opinions provide an accurate overview more quickly, while trusters benefit more from direct experiences in the long-term

[15]. By dynamically adjusting the weighting, the best source of information for the current amount of evidence can be selected, and reasonable trust-aware decisions can be made, while the performed interactions provide more accurate direct experiences. However, it should be noted that the main goal of this step is to accumulate as much representative and statistically accurate evidence as possible. The removal of suspicious or malicious testimony is carried out during the second step of the trust evaluation. [3]

There are numerous different approaches for removing potentially biased third-party testimonies from the collected evidence [3]. For example, the two-way trust model introduced by Alemneh et al. [14] implements a subjective logic-based system to filter out unfair testimony, while DTMS as presented by Chen et al. [16] filters the received opinions by excluding statistical outliers. Other models use intuitionistic fuzzy information [17] or quality of service data [18] to attempt to mitigate the risks of trust-based attacks. Nevertheless, many reputation-based trust evaluation models depend on the unrealistic assumption, that all witnesses share their testimonies at once, to be effective, not considering the fading of past evidence, as well as allowing witnesses to update their opinions after experiencing a change in behavior by the evaluated web application [3].

Analysis As the literature analysis within the scope of this thesis shows, reputation-based trust is currently the most widely researched on approach for trust evaluation. Consequently, there are a lot of modern reputation-based trust models, seven of which are presented and analyzed in the following. The results of the analysis are displayed in table 2.6.

The already mentioned two-way trust management system by Alemneh et al. [14] was introduced in 2019 and is used in fog computing environments. Its trust evaluation combines direct experience evidence with witness testimony to calculate a set of trust metrics that in turn are combined to calculate a trust value. As described previously, this model applies a subjective logic-based system to remove potentially biased or malicious testimony from the aggregated set.

For environments where the privacy of ratings is critical, the self-enforcing trust management system introduced by Azad et al. [19] might provide an appropriate solution. The process of trust evidence aggregation applied by this model is privacy-preserving because trust scores are never publicly visible in plain-text, but instead homomorphically encrypted, which makes it possible to calculate an average of trust scores given to a trustee without having to decrypt them. This way it is not possible to trace back, how a specific truster rated an interaction.

Chen et al. [16] introduced a blockchain-based decentralized trust management system in 2020 which focuses on intelligent transportation environments. The underlying trust model is consensus-based and works together with a blockchain-based trust storage system. Thus, the trust evaluation process is more transparent and witness testimony can be stored irreversibly.

In 2020, Zhai et al. [17] published a trust model for cloud environments that applies intuitionistic fuzzy information based on direct experience data to filter the aggregated witness testimony. The same fuzzy information is later used again to calculate the trust value for each interaction partner candidate and decide for an optimal interaction partner.

The abbreviation Tm-IIoT describes a reputation-based trust model for the industrial IoT that groups nodes within the network into clusters. For each cluster, the most trusted node within the cluster is declared the community leader and is charged with calculating trust values for each of its cluster nodes after receiving necessary observation information from a central control server. Should the another node within a cluster become more trusted than the current community leader, the more trusted node is declared the new leader. Tm-IIoT was published by Boudagdigue et al. [20].

Arifeen et al. [11] describe a trust management model which is used for underwater wireless sensor networks and applies a hidden Markov model to calculate the probability of a node being malicious. This probability can be considered the nodes trust value. While direct observation in the form of the observed packet loss and packet error rate is used for evaluating the trust for continued interactions, witness testimony is used to set up the initial trust values for nodes that are physically moved within the network. The testimony is given by the nodes previous neighbors.

The last analyzed model was published by Rathee et al. [18] and is designed for finding secure rooting paths through mobile information centric networks. To identify malicious nodes, the model analyzes direct observation data like the energy consumption of a device, as well as opinions from neighboring nodes.

2.2.3 Socio-cognitive Trust Models

Socio-cognitive trust models are not evidence-based trust models because, instead of calculating trust according to accumulated evidence of past interactions, socio-cognitive trust evaluations analyze the intrinsic properties of their potential interaction partners, as well as external factors that are likely to allow conclusions about the future behavior of an application [3]. This type of information is derived from a combination of sociological information, mostly acquired through social network analysis – e.g., social relationships and roles of an application, market relationships

Trust Model	R1	R2	R3	R4	R5
Alemneh et al. [14]	$> O(n^2)$	★★★	★★★☆	★★☆☆	★★★
Azad et al. [19]	$O(n)$	★★☆	★★★☆	★★★☆	★☆☆
Chen et al. [16]	-	★★☆	★★★★	★★★☆	★★★
Zhai et al. [17]	-	★★★	★★★★	★★☆☆	★★★
Boudagdigue et al. [20]	-	★☆☆	★★★★	★★★☆	★★★
Arifeen et al. [11]	-	★★★	★★★★	★☆☆☆	★★★
Rathee et al. [18]	$> O(n)$	★★★	★★★★	★★☆☆	★★★
Total	$O(n)$ to $> O(n^2)$	★★★	★★★★	★★☆☆	★★★

Table 2.6: Analysis results for Reputation-based trust models

and dependencies – and stereotypical assumptions or prejudice that base information about an interaction partner on hints that indicate the individual belongs to a larger group – e.g., educational qualifications, previous interaction partners, current or previous workplace [3, 6, 9, 13]. Which specific variables are part of the prejudice analysis depends on the use case, because each environment or network differs [6].

Because of the need for social network analysis, sociological information requires lots of interactions within the network to provide accurate predictions [9]. Prejudice does not and can therefore be used in situations where direct or indirect evidence is not available. However, to properly apply prejudice, an application has to be experienced in executing tasks to be able to assess the influence on trustworthiness by the groups an interaction partner may belong to. [6]

The trust model introduced by Mathas et al. [21] applies a concept of prejudice within an IoT environment by examining which versions of firmware and software a device is running, and if the version is no longer officially supported, has known security vulnerabilities, or fails an integrity check, the device receives a low trust score. Furthermore, the model performs a risk analysis of external factors like a potentially necessary circumvention of network security measures and weighs the trust value against the assumed risks of trusting a device. The same model also uses sociological information based on the observation of a potential interaction partner's not interaction-bound behavior and user-to-user-relationships that transfer trust between the users of a device to its interactions.

Trust Model	R1	R2	R3	R4	R5
Mathas et al. [21]	-	★☆☆	★★☆☆	★★★☆	★★★
DiffTrust [22]	-	★★☆	★★★☆	★★★☆	★★★
Total	-	★☆☆ to ★★☆	★★☆☆ to ★★★☆	★★★☆	★★★

Table 2.7: Analysis results for Socio-Cognitive trust models

Some trust models make use of even more advanced sociological theories. For example, DiffTrust [22] applies the social diffusion theory and sociological data in the form of social proximity within its trust evaluation. Furthermore, DiffTrust also aggregates witness testimony that is later weighted and filtered based on the social proximity of the witness. This means, that this trust model is a hybrid of the groups. However, because of the clear prevalence of sociological information in the trust evaluation, this thesis considers DiffTrust a socio-cognitive trust model.

Analysis Table 2.7 shows the results of an analysis of socio-cognitive trust models on the basis of the model published by Mathas et al. [21] and DiffTrust which was introduced by Fang et al. [22]. The basic functionality of both models is explained in the previous paragraphs.

2.2.4 Organizational Trust Models

Organizational Trust Models are trust models that create and maintain trust by introducing a organizational structure. The applications in this system delegate their trust evaluation to this structure. [3] There are multiple ways this can be achieved. At first glance, this group of models might seem like a bad fit for the decentralized web because of the need for a structural authority, but this is not necessarily the case, as the first example, a trust model based on smart contracts, proves. As described in [1], this model works by using a distributed ledger – similar to the ones used by blockchain technology – to store small parts of executable code. This code contains the contract details for an interaction between two web applications in the network and will be run by nodes of the ledger network according to the conditions set within the contract. This way the two applications do not have to trust each other, but trust in the validity and functionality of the organizational structure – the distributed ledger. In contrast to this, the trust model described in [23] works by generating a central coordination artifact that aggregates trust values from an underlying direct trust-based model and groups the members of the network into roles according to

Trust Model	R1	R2	R3	R4	R5
Ibáñez et al. [1]	-	★★★	★★★★★	-	-
Hermoso et al. [23]	-	★☆☆	★★★★★	★☆☆☆	★★☆
Total	-	★☆☆ to ★★★	★★★★★	★☆☆☆	★★☆

Table 2.8: Analysis results for Organizational trust models

the data provided by its peers. If a web application wants to run a trust evaluation, it will choose the role that is best suited to provide to a successful interaction and pick one of its members. The roles are maintained and regularly re-evaluated by the coordination artifact.

Analysis The analysis for this group of models has been conducted based on the previously explained organizational trust models by Ibáñez et al. [1] and Hermoso et al. [23]. The results of this analysis are displayed in table 2.8.

2.3 Comparison

After presenting the different groups of approaches for computational trust evaluation, as well as their representatives, in chapter 2.2, they are now to be compared against each other in regard to the requirements formulated in chapter 2.1. In table 2.9, this comparison is displayed. The following abbreviations are used to conserve space:

Dir = Direct Trust-based trust models
Rep = Reputation-based trust models
SC = Socio-Cognitive trust models
Org = Organizational trust models

It can be observed that direct trust-based trust models, as well as reputation-based ones, are more decentralized than socio-cognitive and organizational ones. However, the analyzed organizational trust models show a strong variation regarding requirement R2. Furthermore, the trust evaluation time complexity could not be determined for two out of the four groups. This is because authors often present their trust models in an abstract manner and rarely specify implementation details or descriptions of

	Requirements				
	R1	**R2**	**R3**	**R4**	**R5**
Dir	$O(n)$	★★★	★★★☆ to ★★★★★	★☆☆☆	★★★
Rep	$O(n)$ to $> O(n^2)$	★★★	★★★★	★★☆☆	★★★
SC	-	★☆☆ to ★★☆	★★☆☆ to ★★★☆	★★★☆	★★★
Org	-	★☆☆ to ★★★	★★★★	★☆☆☆	★★☆

Table 2.9: Evaluation matrix for all groups (see chapter 2.2) and requirements R1-R5 (see chapter 2.1)

the algorithm flow in the form of pseudocode. It also becomes apparent that most of the presented approaches display insufficient situational awareness. As expected by their group definition, socio-cognitive trust models conduct more extensive analyses of the interaction context, yet still lack contentual considerations during their trust evaluation.

Two of the posed requirements have been fulfilled well by all presented approaches. Firstly, most of the analyzed approaches do not require human help to function, except for the socio-cognitive approaches, which at least had to be set up manually. Secondly, the used trust scales are able to express trust on a spectrum and allow for nuanced trust-aware decisions.

2.4 Conclusions

The examined trust models show that even nowadays, entity trust is still the prevalent trust management solution. As expected, the situational awareness of the analyzed trust models is low and none of the analyzed models consider the contentual information of an interaction during the trust evaluation. However, the importance of decentralization and working without requiring humans as trust fallback mechanism were already recognized by most trust models, and implemented well. This means none of the presented approaches is appropriate for the use case that was portrayed in chapter 1.2, but they provide a solid basis for a content trust-based trust model.

The literature review also showed that there are a number of important aspects called *trust dimensions* that a well-designed trust model needs to consider [6]. While there are numerous more, there are four trust dimensions that shall be discussed in greater detail to be used later in the concept for this thesis' problem solution. The four trust

dimensions that will be discussed are: *Trust Semantics*, *Trust Preferences*, *Trust Delegation*, and *Initial Trust*.

Trust Semantics Trust semantics are used for interpreting calculated trust measurements [24, 25]. They are often specified as ontologies or numerical values, and are necessary because most trust models use only one value which indicates whether a trustee is reliable [6]. The interpretation of this value is not obvious in most contexts, which makes further clarification necessary [24]. This clarification is given by turning this singular value into a composite value that embodies the average of multiple sub-aspects [6]. For example, consider an e-commerce platform where buyers rate sellers. Instead of having the platform provide a potential buyer with an ominous trust score, the trust value may be an average of quality of service attributes like product quality, delivery time, and accuracy of the product description. These separate aspect can be considered and weighted differently to adjust the calculation of the final trust value. [24, 25]

Trust Preferences Trust is never absolute and highly use case specific, which means that different web applications very well may have different preferences for when to consider a service reliable or trustworthy. This idea is summarized as *personalized trust rate*, which describes the situation that different trusters require different trust values for the same trustee. The web is an open, globalized system that spans different countries, cultures, and regulations. Consequently, the expectations in trust management differ depending on the situation and use-case of a web application. Thus, trust models should provide a way to specify individual trust preferences to allow bridging over cultural differences of trust. [25]

Trust preferences work well together with trust semantics, since trust preferences can be used as the weights of the different sub-aspects of a trust value [6]. To revisit the previous example, trust preferences could be used to allow a potential buyer to adapt the ratings of sellers according to their current requirements, and e.g., sacrifice the need for a short delivery time in favor of a high product quality.

Trust preferences are often managed internally by the web application. To adapt the trust evaluation to the needs of an application, a mechanism for merging system priorities with application preferences is necessary. Also, by using common ontologies, trust preferences can be shared and reused easily. [6]

Trust Delegation Trust delegation describes that a truster trusts a trustee to make decisions about a resource or service the truster controls on their behalf, based on delegation schemes the truster created. This is an example for trust transitivity.

Trust transitivity means that if there are two entities A and B that trust each other, every entity C that B trusts, is also trusted by A, even if A has never met C. This is a controversial topic in the scientific community, since trust transitivity can lead to unexpected or adverse results for the truster. Therefore, if a trust model uses trust transitivity, mechanisms for preventing and mitigating undesired side effects have to be included. [26]

Initial Trust Trust has three development stages [6, 25]:

1. Building trust or setting trust up initially
2. Stabilizing trust
3. Dissolution of trust over time

Most trust models assume that trust already exists or is already initialized in some way, but it is critically important that models properly initialize trust values for web applications that newly joined the web [25]. There are two reasons for this [6]: (1) new web applications must not be restrained by a lack of interaction evidence and reputation, but (2) existing web applications must not be cheated by newcomers.

Granatyr et al. [6] describe the setting up of initial trust as a major difficulty when developing new trust models. However, there is a plethora of available solutions to this problem, where some work based on machine learning or prejudice in situations without evidence, while others use a trust-distrust trust scale with newcomers being placed in a neutral position in-between both extremes.

3 Concept

In this chapter, the concept for the following implementation is presented. Chapter 3.1 contains the general idea for the trust framework that will be implemented later, as well as an explanation of the calculation of the final trust value. As explained in problem analysis of the introductory chapter 1.3, the main objective of this thesis is to determine, how and to what extent each factor of content trust, as described by Gil et al. [5], can be implemented and used for trust evaluation by web applications. Each factor of content trust will therefore be described in chapter 3.2 and its conceptualized implementation explained.

The trust framework that is built over the course of this thesis will from this point on be called and referenced as CRAB, which stands for **C**ontent t**R**ust ev**A**luation model for the distri**B**uted web.

3.1 Process Overview and Trust Value Calculation

The final trust value calculated by CRAB is a composite value that is based on multiple sub-aspects of content trust. These aspects are a subset of the content trust factors described in [5] and are used to produce values that are aggregated with a weighted average approach. This process implements two of the trust dimensions described in chapter 2.4: trust semantics and trust preferences. While trust preferences are applied in a spread manner to allow for fine-tuning of the intricacies of the trust evaluation process, they mainly contribute the weights needed for calculating the average value of the subset of content trust factors that represent trust semantics. This enables the use of use-case specific modes of operation, as well as allowing for a deep understanding and easy interpretation of the final trust value.

The equation that is used to calculate the final trust value is displayed in equation 3.1, where R describes the resource that trust is evaluated for and c represents the number of content trust factors. The amount of used trust factors varies depending on the trust preferences and is therefore adjusted dynamically. The variable f stands for a trust factor, with $f_{R,k}$ describing the k-th trust factor that is evaluated for the resource R, with index k being adjusted dynamically according to c as well. Finally, w stands for the weight of the given trust factor and v embodies its value. There are

some exceptions where the calculation will be skipped or adapted according to the current situation, this is explained in chapter 3.2.

$$final\,trust\,value(R) = \frac{\sum_{n=1}^{c}(w(f_{R,n}) * v(f_{R,n}))}{\sum_{n=1}^{c} w(f_{R,n})} \tag{3.1}$$

As highlighted in chapter 2.4, a necessary requirement that trust models have to fulfill, is the initialization of trust, also called *Initial Trust*. But, before this can be discussed, a trust scale has to be specified first. CRAB implements the trust scale introduced by Marsh and Briggs [8]. This means the trust value will exist in a $[-1, 1]$ interval, where -1 represents the largest possible amount of distrust and 1 stands for largest possible amount of trust. A trust value of 0 describes trust-related ignorance towards a resource. The final part of the trust scale is the cooperation threshold, which can theoretically be placed anywhere in the described interval. As already explained in the introductory chapter 1.3, the cooperation threshold is the minimum trust value a resource requires to be interacted with. The cooperation threshold is dynamically adjusted based on the criticality of the current situation (see content trust factor *Context and Criticality* in chapter 3.2). If the cooperation threshold is above 0, Marsh and Briggs designate the interval $[0, cooperationthreshold]$ as *Untrust*. If the trust value of a resource falls within this interval, it means the resource is indeed technically trusted – yet it is not trusted enough for the current interaction.

This thesis does not discuss concepts or implementations for deriving initial trust. Thus, both the concept itself, as well as the matching implementation presented in chapter 4 will not make assumptions towards initial trust. Therefore, if the trust value for a content trust factor cannot be calculated, possibly due to e.g., missing data from the resource or the trust preferences, the factor will be ignored during the final trust value calculation.

In figure 3.1, a possible layout for the entire *Content Trust Awareness* subsystem is displayed. This thesis focuses exclusively on the trust evaluation component. The functionality of the other components, as well as their dependencies, are beyond the scope outlined in the introduction and therefore not addressed. Due to space limitations, the interface labels are not included in the figure, but instead replaced by indices which can be referred to in table 3.1.

The trust evaluation process as described in this chapter, is represented within the figure by the *Trust Evaluation* component. The interfaces illustrate the necessary data flow within the subsystem, with the remaining components showing a possible structure to provide and receive the data for and from the trust evaluation.

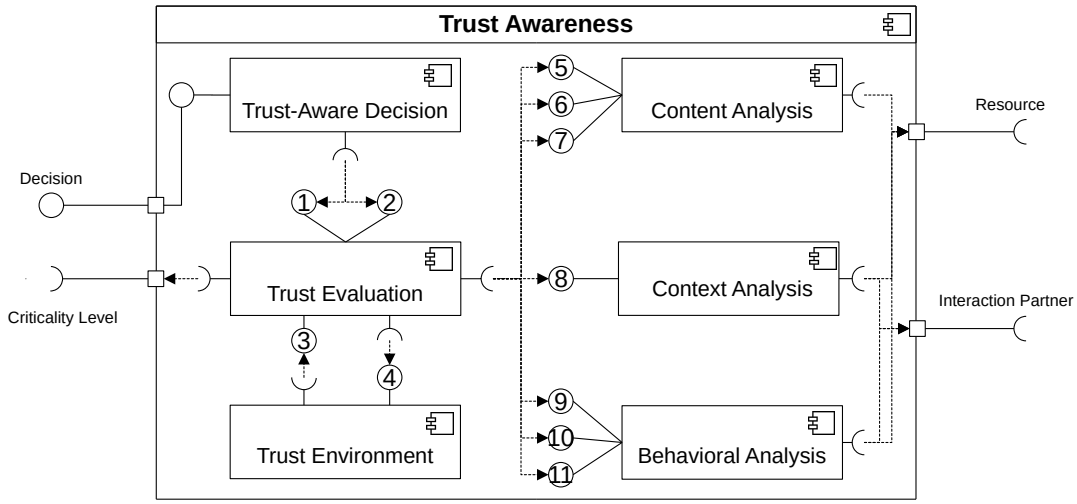

Figure 3.1: Component Diagram of the proposed Trust Awareness subsystem

Index	**Description**
1	Trust Value
2	Cooperation Threshold
3	Latest Evaluation Result
4	Trust Environment Data
5	Parsed Resource Data
6	*Likelihood* Value
7	*Specificity* Value
8	Parsed Interaction Partner Data
9	*Bias* Value
10	*Deception* Value
12	*Incentive* Value

Table 3.1: Indices for Figure 3.1

As displayed in figure 3.1 and described in table 3.1, *Trust Environment Data* is a required interface for the trust evaluation. It represents a data structure containing recommendations from other web applications regarding the evaluated resource, as well as the evaluation history and trust preferences. This is considered environment data from the current web application and the network the web application is participating in.

Evaluation History The evaluation history is a database that contains the results for past trust evaluations. It should be noted, that not only the final trust value is saved in this structure, but also the values for every implemented content trust factor, as well as the necessary data for identifying and potentially contacting the web application that provided the resource. Every entry further contains the corresponding resource identifier. Depending on the size of the network and the amount of performed trust evaluations, the amount of data may grow rapidly. To mitigate this, the schema of the database can alternatively be changed to contain the average evaluation results for each interaction partner that provided a requested resource. The effect of this change will depend on the network, however, for the portrayed scenario of a decentralized web, it is expected that the number of information providers is significantly smaller than the number of available resources, which means, by only keeping one data set per interaction partner, less data has to be saved. But this approach introduces new functional disadvantages, e.g., the usage of *Direct Experience* and *Recommendation* as content trust factors will be less context-sensitive, because they cannot be applied individually for each resource and a resource's references could not be evaluated as described in chapter 3.2. Due to these disadvantages, CRAB will use the former variant – an evaluation history with exactly one data set per evaluated resource.

Trust Preferences In chapter 2.4, the need for a customizable trust evaluation process is explained. To fulfill this need, CRAB implements a simple configuration system which allows a web application – or its owner – to decide beforehand, which content factors should be used within the evaluation and how they should be weighted towards each other. This configuration also allows to specify the mode of operation and the assessment standard of some factors. How this affects each factor in particular is explained in the following chapter 3.2.

3.2 Content Trust Factor Implementations

This chapter highlights how each content trust factor, as presented and defined by Gil et al. [5], will be implemented in CRAB. Figure 3.2 shows a BPMN diagram of the trust evaluation process and will be used in the following to explain this process.

As shown by the starting event *Resource received*, the trust evaluation starts after a resource has been sent by an interaction partner and was parsed by the underlying framework. This data, as well as information about the interaction partner, is represented as *Resource Data* and *Interaction Partner Data* respectively.

Context and Criticality The task *Recognize Criticality Level* represents the second factor of content trust, *Context and Criticality*. It describes, that the context of a trust-aware decision, as well as the severity of its consequences, influences the perception of trustworthiness in a resource. This factor is not implemented as a separate measurement, but changes the cooperation threshold of the interaction based on the Criticality parameter provided during the trust evaluation call and the corresponding values for each level of Criticality, as configured in the trust preferences. The calculated value for the cooperation threshold is then provided as output for the trust-aware decision, hence excluding lowly trusted resources from interactions in critical contexts.

Recency Trust and the resources it is associated with change over time and as such should be re-evaluated after a certain time [5]. This thought is represented by the content trust factor *Recency*. However, most aspects of content trust are dynamic and are therefore re-evaluated automatically with each trust evaluation. The only parts of the evaluation process that are affected by obsolescence are the local and third-party evaluation history. To prevent the usage of obsolete data, *Recency* is implemented by applying an age filter which simulates "forgetting" evaluation results after a certain time. If a task requires the evaluation history, the age filter will be applied before the data is transmitted. The maximum age for interaction entries is defined within the trust preferences.

Topic The content trust factor *Topic* describes that the trustworthiness of a resource can depend on its topics, as some resources or agents may only be trusted on certain topics. This is implemented in CRAB as shown by the *Calculate Topic Value* task and works based on data derived from the evaluation history. As mentioned in the beginning of this chapter, after each evaluation the respective results are saved in a database. This includes, among other things, the topics for each past interaction

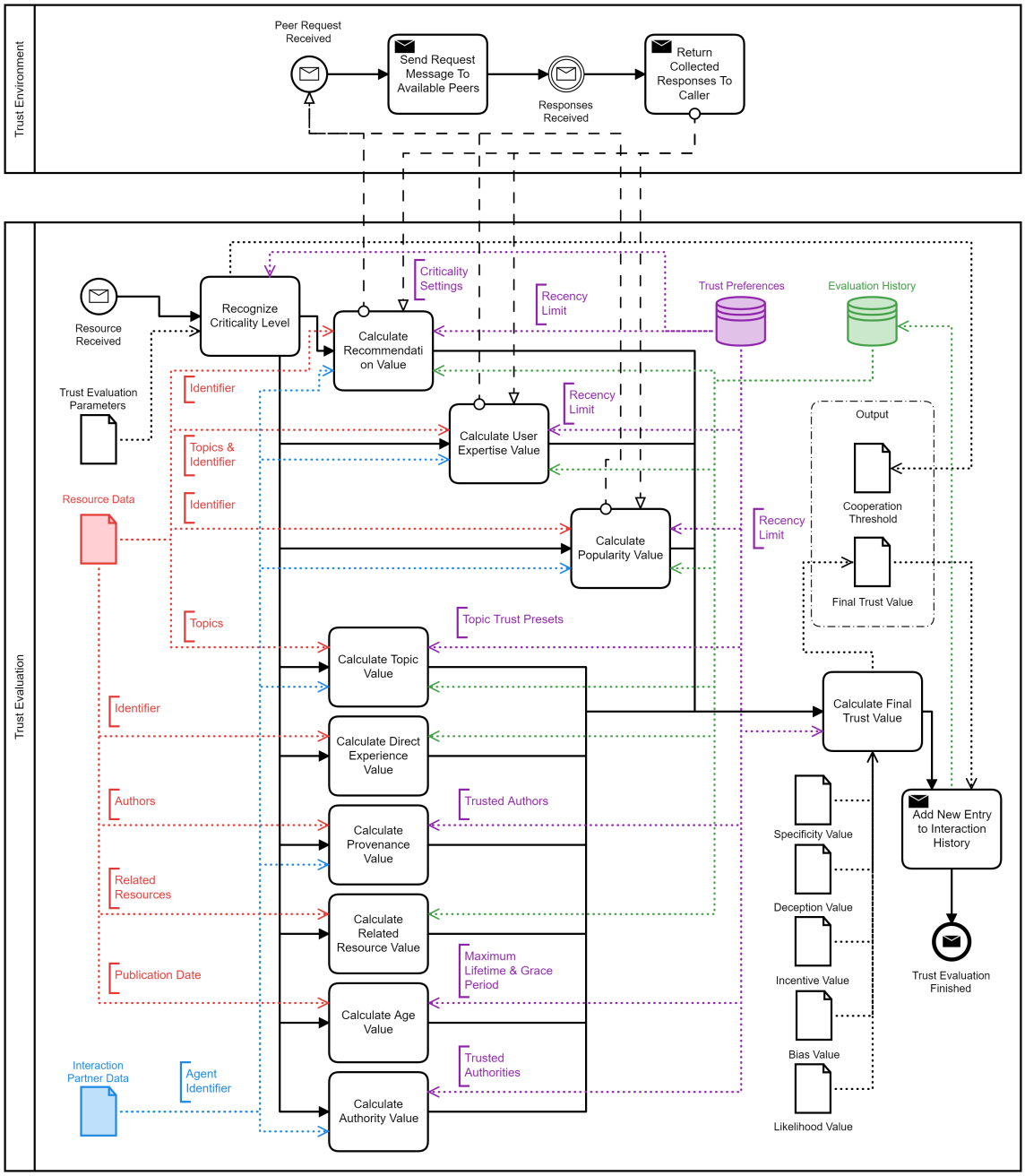

Figure 3.2: BPMN diagram of the trust evaluation process in CRAB

and the resulting final trust score. After an evaluation has been saved, a future trust evaluation for a resource with similar topics will collect these entries and calculate the average trust score. This value is the *Topic* value for the resource. In cases

where a resource contains multiple topics, those topics are stored separately with the same final trust value associated with them. This allows future trust evaluations to combine these entries freely. For example, consider a situation having two past evaluations with the first one being about a resource containing the topics 'A' and 'B', and the second one being about a resource containing the topics 'B' and 'C'. Now, a third evaluation is being performed for a resource containing only topic 'B'. In this case, both previous evaluation results will be used to calculate the *Topic* value.

In environments with only a small number of interactions and evaluations, there might not be sufficient entries in the evaluation history to cover each topic. To allow a limited calculation of *Topic* trust nevertheless, it is possible for agents to predefine *Topic* trust values for specific agents in its trust preferences. This value will be added to the calculation of the *Topic* value with the same weight as an entry in the evaluation history. Therefore, the predefined *Topic* trust values will greatly lose significance as soon as more evaluation results are created.

It should be noted, that CRAB expects the content analysis component to correctly detect all topics that are contained in the resource. Alternatively, should the detection of topics with absolute certainty not be possible, each topic could be provided and later be stored with an associated confidence value by which the respective trust score saved in the evaluation history can be weighted for calculating the *Topic* value as weighted average instead.

Popularity The popularity of a resource may hint towards its trustworthiness and is therefore analyzed by the third factor of content trust. However, the exact implementation of the calculation of the *Popularity* value, as represented by the task *Calculate Popularity Value*, may differ. For the prototype of CRAB presented in chapter 4, the popularity of a resource will be derived from a heavily simplified, machine-accessible social network. For each resource that is being evaluated, the evaluating web application is sending requests to its peer group to find out how popular the resource is in a group of applications that are assumed to have similar requirements and cultural imprints as the evaluating application. Each peer then checks its evaluation history for the specified resource and returns a Boolean value that signals whether its history contains at least one entry for this resource that resulted in a trust value above the peer's cooperation threshold and does not exceed the *Recency* limit allowed by the requester. The peer responds with *true* if those conditions are met, otherwise with *false*. After collecting the messages from its peers, the evaluating web application calculates the *Popularity* value as the proportion of peers that responded with *true*. The resulting *Popularity* value will be within the interval $[0, 1]$ to ensure that resources are not penalized too heavily for their lack of popularity, as it is not necessarily an indication of high information quality. It should be noted further, that it does not

matter whether a peer actually interacted with the resource, only the final result of the trust evaluation is important in this step.

Gil et al. [5] use a version of the PageRank algorithm to determine resource popularity. However, this method ignores the context in which a resource is embedded or referenced and therefore may be harmful – e.g., because someone may embed a link to the resource on their website with a warning about the resource's malicious intentions, but PageRank would treat this the same way as a recommendation to visit the link.

Authority Resources shared by trusted authorities (e.g., newspapers, government agencies) will often be more trustworthy than personal blogs and should therefore be rated higher when evaluating trust. This is formulated by the fourth factor of content trust, *Authority*. To determine whether a resource was shared by an authority the web application trusts, the task *Calculate Authority Value* receives an identifier from the interaction input, as well as a list of trusted authorities that have been configured into the trust preferences. If the interaction partner is considered a trusted authority, the *Authority* value is set to 1, otherwise the factor is omitted from the final trust value calculation.

An alternative to using a simple *Authority* value to affect the final trust value would be an implementation that ensures that authorities are always cooperated with, even if the values produced by other content trust factors are low, by assigning a final trust value above the cooperation threshold, even if the one calculated was well below the threshold. This would allow for interactions in low trust environments where none of the provided resources reach above the cooperation threshold. However, this method is not used because it would undermine key aspects of content trust, such as the separation of trust associated with an entity from trust associated with a resource [5]. The basic principle of content trust is a dynamic analysis of contentual and contextual properties of a resource to ensure reliable trust-aware decisions in dynamic environments that may change arbitrarily. Statically trusting entities that are considered authorities by the web application hurts this principle, and can be dangerous in situations where the authority has been corrupted or impersonated by malicious third-party actors.

Direct Experience *Direct Experience* is a pretty straightforward content trust factor that is derived from the local evaluation history. This history contains the final resource trust values that have been calculated in previous trust evaluations, regardless of whether an interaction happened or not. The task *Calculate Direct Experience*

Value receives this evaluation history, calculates the average of the received entries and emits the result as the resource's *Direct Experience* value.

Recommendation Similar to previously described reputation-based trust models, *Recommendation* describes a mechanism which aggregates and analyzes third-party testimony to derive a statement about the resource's trustworthiness. In CRAB, this is implemented by the task *Calculate Recommendation Value*, which receives witness testimony based on the evaluation history of other participants of the network. Each witness forms its testimony by calculating the mean of the final trust values for the currently evaluated resource. This testimony is filtered by weighting each response with the trust value for the respective agent (this weight is calculated similarly to the testimony itself). If the trust value for an agent is below the current cooperation threshold, their testimony is ignored. To calculate the *Recommendation* value, the median of all received testimony is used. This approach helps to mitigate the effect of testimony from non-representative outliers on the final *Recommendation* value.

Related Resources The content trust factor *Related Resources* describes that the trustworthiness of a resource is influenced by the given references. This content trust factor is flawed because giving a reference to a high quality resource does not guarantee a high information quality or trustworthiness of the referencing resource. However, resources that cite sources that are deemed reputable are considered more trustworthy than resources that are based on dubious or no references at all. The corresponding *Related Resources* value is calculated during the task *Calculate Related Resource Value* and is based on a list of referenced resources provided by the *Resource Data* that are then evaluated based on the evaluation history to determine which references should be considered reputable.

Provenance The content trust factor *Provenance* is based on the belief that the trustworthiness of an author may be transferred onto resources created by them. This is implemented within the task *Calculate Provenance Value* and works similarly to the calculation of the authority value. First, the task receives identifiers for the original resource authors [1], as well as a list of configured trusted sources. If one or more of the resource authors are trusted, the *Provenance* value is set to 1, otherwise it is kept undefined and the factor will be ignored in the final trust value calculation.

[1] It should be noted that the interaction partner may not be the original resource author, but instead e.g. an information aggregator or social network component that hosts the requested resource.

User Expertise Topic experts may identify trustworthy resources within their area of expertise more reliably than non-experts. This assumption is constituted by the *User expertise* content trust factor. It is implemented within the task *Calculate User Expertise Value* and aims at aggregating expert testimony about the evaluated resource. To achieve this, the task requires the topics and the identifier of the resource, additionally to the available witness testimony local evaluation history. During the next step the resources with similar topics to the one currently evaluated are chosen, if their trust values are above the current cooperation threshold. For these trusted resources, the saved original authors are gathered and their witness testimony regarding the current resource is collected. It should be noted that authors are not necessarily entities in the network, especially since human authors do not have a direct interface to the network. For this reason, the web application that first made the resource available on the network is considered its original author and will be asked for expert testimony on the currently evaluated resource. There may be multiple authors for a given resource if the web application that first provided the resource specifies them as co-authors. Finally, the *User expertise* value will be output as the arithmetic mean of the received expert opinions.

Age Not only trust can deteriorate over time, but also the validity and information quality of resources. For this reason, the task *Calculate Age Value* tests, whether the resource age is within the configured maximum resource lifespan. If so, the *Age* value is equal to 1. However, should the resource age exceed the maximum lifespan, a configured *grace* timespan is applied. If x represents the number of seconds by which the maximum lifetime has been exceeded and the grace timespan is given in seconds, the *Age* value is calculated as $max(1 - \frac{grace}{x}, 0)$.

Following this, there are two possible ways to handle the excess of the maximum lifespan: (1) the interaction is declined and the trust evaluation result is set to full distrust if $Age < 1$ because the information within the resource has lost its validity (e.g., weather reports for the current week are not valid anymore two weeks later), or (2) the information is not time-dependent and the information contained in older resources is not necessarily false or invalid, however more recent resources are assumed to be more trustworthy than older ones. To reflect this in the trust evaluation, the *Age* value is added to the calculation of the final trust value. Which method is preferred depends on the use case and therefore needs to be configured within the trust preferences.

3.2.1 Externally Calculated Content Trust Factors

The following content trust factors are not directly calculated within CRAB itself based on the received data, but provided by other components of the *Trust Awareness* system (see figure 3.1 in combination with table 3.1) and used in the final trust value calculation. For this reason, all provided content trust factor values have to be mapped to the trust scale by Marsh and Briggs [8]. Thus, if e.g., the *Bias* value equals 1, it means that no bias was detected, as it represents the highest possible trust value according to the used trust scale.

Bias If a source is biased, it may lead to the source providing misleading or false information to help further its vested interest. Bias is in most cases hard to detect because of its frequent subtlety and the deep subject understanding that is needed to recognize it. As such, the *Bias* value required by final trust value calculation is provided by the *Behavioral Analysis* component and directly injected into the final calculation.

Incentive In some situations, a source might have a motivation to provide accurate, high quality information, which in turn might makes the resource more trustworthy. This situation is rewarded by the content trust factor *Incentive*. Similarly to the *Bias* value, the *Incentive* value is not calculated during the trust evaluation and instead received from the *Behavioral Analysis* component.

Deception Contrary to *Bias*, which might be unintentional, deception describes a situation where the content or metadata of a resource are false as the result of active lying and are meant to deceive the evaluating web application. This is reflected by the content trust factor *Deception*. If an attempted deception is detected, instead of using the *Deception* value during the final trust value calculation, the calculation is skipped and the emitted trust value is set to full distrust, i.e. −1 on the Marsh and Briggs trust scale. A resource is deemed deceptive if the *Deception* value provided by the *Behavioral Analysis* component is below the *Deception* threshold configured in the agent preferences.

Specificity This content trust factor is based on the assumption that a resource might be more trustworthy if it is precise and specific. The corresponding *Specificity* value, again, is not calculated by the trust evaluation, but provided by the *Style Guide Check* component (see figure 3.1). While the implementation of this component will

not be addressed in this thesis, the idea of enforcing a style guide to determine the content specificity originates from [27].

Likelihood If the web application already has knowledge about the topics of a resource, it may be able to make statements about its trustworthiness based on the laws and limitations of the topics. This requires a deep content analysis together with the already available knowledge base of the web application. The calculation of the corresponding *Likelihood* value is done by the *Content Analysis* component (see figure 3.1) and, similarly to other content trust factor values that are calculated by external components, injected directly into the final trust value calculation.

3.2.2 Unused Content Trust Parameters

Based on the use case context of web applications, not every content trust factor was implemented in CRAB. Three factors were excepted from the trust evaluation. They are described below, including the reason for their exclusion.

Limited Resources If there are not sufficient alternatives to an evaluated resource for the current topic, an application may cooperate with less trustworthy information. However, this is not part of the trust evaluation, but instead part of the trust-aware decision, and as such outside the scope of this thesis and the current version of CRAB.

Agreement This factor describes that a resource which might not be trustworthy on its own, gains trust, if a large number of other resources agree with its information. *Agreement* is part of the trust-aware decision as well, and therefore outside the scope. Furthermore, the assumption that the majority is correct, may be questionable and the usefulness of the factor potentially low.

Appearance The appearance of a resource may affect an user's perceived trustworthiness of this resource. This factor is not implemented because the appearance of a resource does not matter for the communication between machines.

4 Implementation

This chapter describes an implementation of the concept that was specified in chapter 3. CRAB's prototype implementation is embedded in the trust testbed aTLAS [4] and uses its interfaces to simulate trust evaluation in a multi-agent system. The source code for aTLAS, an explanation of its objectives and an in-depth description of the trust testbed's inner workings can be found via the official project page[1]. The source code for CRAB's prototype implementation is accessible via the *content_trust* branch in the respective repositories for the aTLAS *Trustlab*[2] and *Trustlab Host*[3]. It should be noted that aTLAS is under active development and not yet fully feature complete.

As explained in the introduction, a decentralized web is similar to an open, dynamic multi-agent system. As such, web applications of this decentralized web are similar to agents in a multi-agent system. Thus, the terms agent and web application will be used interchangeably in the context of the trust evaluation explained in this and the following chapters.

In the following sections, the functionality of the implementation of the trust evaluation process is illustrated based on excerpts of the source code, which are written in Python. The line numbers given for each listing are not equivalent to the original source code.

4.1 Basic Functionality and Evaluation Flow

The trust evaluation process is started by a call to the *eval_trust* function, which initializes and orchestrates the control flow of the trust value calculation. This function call is performed by aTLAS during the truster agent message loop, which handles incoming messages for an agent. During this call, aTLAS provides an interface to the trust model containing the following set of input parameters:

[1] `https://vsr.informatik.tu-chemnitz.de/projects/2020/atlas` [21.12.2021]
[2] `https://gitlab.hrz.tu-chemnitz.de/vsr/phd/siegert/trustlab/-/tree/content_trust` [21.12.2021]
[3] `https://gitlab.hrz.tu-chemnitz.de/vsr/phd/siegert/trustlab_host/-/tree/content_trust` [21.12.2021]

- the unique identifiers for the agent that is calculating the trust value for the incoming resource (*agent*) and the agent that provided it (*other_agent*)
- an Observation object that contains the resource, as well as metadata for the transmission and the resource (*observation*)
- a dictionary data structure which contains the trust preferences used by the current agent (*agent_behavior*)
- a Scale object that provides an interface to address any arbitrary number interval-based trust scale used by the agent (*scale*)
- a Logger object that provides an interface to read and append to the evaluation history for the agent (*logger*)
- a dictionary data structure which contains the addresses of all agents (*discovery*)

The text written in brackets contains the variable name that is consistently used for the corresponding object throughout the implementation.

The contents of the Discovery dictionary should be context and agent specific. For the current aTLAS implementation, it holds the contact information (i.e. IPv4 address and port) of every agent in the scenario. However, aTLAS scenarios usually simulate small networks. For larger networks – e.g., the decentralized web – it would be more efficient to only store an agent-specific peer group of agents. How this peer group is chosen should be determined by the respective implementation, but for example a selection by areas of interest or physical location would be conceivable.

Listing 4.1 shows, how different content trust factors are used in the main evaluation function. Starting with *Context and Criticality*, which changes the cooperation threshold of the scale in accordance with the trust preferences configured for the agent. To ensure the Criticality level was set for the current message and the agent has configured respective cooperation threshold values, line 1 checks for the existence of the keys in the observation object and trust preferences respectively. Only if both keys exist, the new value for the cooperation threshold is extracted from the *content_trust.context_values* dictionary, using the Criticality string as addressing key, and sent to the current scale object to change the threshold accordingly.

The next code block shows the usual procedure for the usage of a content trust factor. First, the trust preferences are searched for the key for the current factor – as shown in line 6 for *content_trust.recommendation* – to determine whether this factor was enabled by the agent. Should that be the case, the respective value is either calculated, as shown for the factor *Recommendation* in line 7, or extracted from the observation object to simulate the surrounding analysis components of the

trust awareness subsystem. This can be observed exemplarily for the content trust factor *Bias* in line 12. After calculating or receiving a trust factor value, it is added to the evaluation history via the logger interface. As the call to the logger function in lines 8 and 13 show, the value is saved in combination with the identifiers for the current agent and interaction partner, the name of the respective content trust factor and the unique identifier for the evaluated resource. The current time is added to each entry as well, to provide timestamps for the filtering of evaluation history entries to enforce the *Recency* trust factor. Saving all this data for each content trust factor ensures, that past evaluations can be used flexibly for future trust value calculations. Finally, the factor value is added to the *trust_values* dictionary for the final trust value calculation, with the names of the factors as keys to assign the weights as configured in the trust preferences.

```
1  if 'content_trust.context_level' in observation.details and ↙
       ↳ 'content_trust.context_values' in agent_behavior:
2      context_level = ↙
       ↳ observation.details['content_trust.context_level']
3      cooperation_threshold = ↙
       ↳ agent_behavior['content_trust.context_values'][context_level]
4      scale.set_cooperation_threshold(cooperation_threshold)
5
6  if 'content_trust.recommendation' in agent_behavior:
7      recommendation_value = content_trust_recommendation(agent, ↙
       ↳ other_agent, resource_id, scale, logger, discovery, ↙
       ↳ recency_limit)
8      logger.write_to_agent_trust_log(agent, ↙
       ↳ 'content_trust.recommendation', other_agent, ↙
       ↳ recommendation_value, resource_id)
9      trust_values['content_trust.recommendation'] = ↙
       ↳ recommendation_value
10
11 if 'content_trust.bias' in observation.details:
12     bias_value = observation.details['content_trust.bias']
13     logger.write_to_agent_trust_log(agent, 'content_trust.bias', ↙
       ↳ other_agent, bias_value, resource_id)
14     trust_values['content_trust.bias'] = bias_value
```

List of Listings 4.1: Calling content trust factor sub-functions

Displayed in listing 4.2 is the preparation of the *trust_values* dictionary for the final trust value calculation and the coordination flow for the call to the weighted average function. The first step is the removal of content trust factor values that have been added with their respective key but are equal to *None*, which means they exist, but are undefined. There are multiple situations that might lead to this. For example, if the agent that provided the evaluated resource is not an authority recognized by the truster agent, the *Authority* value is set to *None* and added to the *trust_values* dictionary. To ensure that only valid values are forwarded to the

final trust value calculation, all undefined values have to be removed. This is shown in line 1. Subsequently, the variable *final_trust_value* is introduced in line 2 and assigned the default value of the currently used scale in case the agent has not properly defined a valid calculation method for the final trust value. It is expected that aTLAS will support different methods for calculating the final trust value in the future. Therefore, in lines 3 and 4, flow control for deciding the appropriate calculation method is added. In line 5, the weighted average function is called with the calculated trust values, weights per content trust factor and default value for the current scale as required parameters. The scale default value is needed in case parts of the calculation go wrong and a fallback value is needed. Line 7 marks the final line of the trust evaluation function and contains a return call, which provides the calculated final trust value back to aTLAS, where it can be used for the trust aware decision. The trust scale does not have to be exported explicitly, as it was provided as function parameter and edited in-place, so the new cooperation threshold is also accessible from the function caller.

```
1 trust_values = {metric: value for metric, value in ↙
    ↳ trust_values.items() if value is not None}
2 final_trust_value = None
3 if agent_behavior['__final__']:
4     if agent_behavior['__final__']['name'] == 'weighted_average':
5         final_trust_value = weighted_avg_final_trust(trust_values, ↙
    ↳ agent_behavior['__final__']['weights'], None)
6 return final_trust_value
```

List of Listings 4.2: Coordination of the final trust value calculation and return call of the trust evaluation function

This describes the overall structure of the trust evaluation process. The next section will introduce a few exemplary implementations for content trust factors to show how the calculation of these values and the communication between different agents in the network works.

4.2 Implementations of the Content Trust Factors

The calculation of content trust factors takes place in the corresponding functions. Some factors are calculated without communicating to other agents, while other factors work by requesting witness testimony or popularity data by third-party agents and deriving this factor's value from it. Implementations for both types factors, as well as the implementation for the *Direct Experience* factor will be shown and explained in this chapter. These implementations present the basic calculation mechanisms for content trust factors, as well as giving an overview over the communication

between agents that is taking place during the execution of the trust evaluation process implemented in CRAB and the interaction with the evaluation history. These code listings are meant to be examples and therefore do not include all factors implemented in CRAB and might be missing essential code parts (e.g., import statements) that are omitted from the listings for clarity.

An example for a content trust factor that works without inter-agent communication is *Provenance*. Its implementation is shown in listing 4.3. The function requires three parameters. In display order, these are: (1) the set of resource authors extracted from the resource itself, (2) the set of trusted authors configured in the agent's trust preferences, and (3) the scale used by the truster agent. As lines 2 and 3 show, the factor value will only be calculated, if at least one resource author was detected. Should that not be the case, the *Provenance* value is left undefined. Should the resource authors be known however, the function determines the intersection of the *authors* set and the *trusted_authors* set in line 5. This intersection is assigned an intersection score in line 6, which equals the proportion of resource authors that are trusted. Finally, this score is mapped to the scale interval and returned to the caller function in the last line.

```
1 def provenance(authors, trusted_authors, scale):
2     if len(authors) == 0:
3         return None
4
5     count_congruent = len(set(authors) & set(trusted_authors))
6     score = count_congruent / len(authors)
7     return scale.normalize_value_to_scale(score, 0, 1)
```

List of Listings 4.3: Calculation of the *Provenance* value

Listing 4.4 shows two functions of which only one is used for calculating the *Direct Experience* value. This is done by the *direct_experience* function that interacts with the evaluation history database to determine a *Direct Experience* value. The function begins by requesting all of its historic evaluation data from the logger object which represents the evaluation history database. In lines 5 to 7, the received history data is filtered using Python's list comprehension system to extract the final trust values for each history entry for the currently evaluated resource. These entries are detected using the identifier of the resource given by the function parameter *resource_id*. Furthermore, they are filtered to only include entries that have been created within the maximum allowed lifetime represented by the *recency_limit* to allow old evaluation results to be forgotten after the desired timespan. The latter being the implementation of the *Recency* factor, as mentioned above. The final check `entry['trust_value'] != 'None'` is used to remove entries for which the trust value calculation failed. The calculation of the *Direct Experience* value is completed

in lines 10 and 11 by calculating the mean of the given history values, if at least one such entry exists, or returning the *None* otherwise.

Normally, CRAB calculates the final trust value for a resource opposed to the sender of the resource. This is shown in the first function, where the history entries are filtered solely based on the identifier of the currently evaluated resource. However, the second function within this listing provides a helper function for weighting user testimony for the calculation of the *Recommendation* value, as explained later in this chapter. The basic functionality of this function, whose definition begins in line 12, is similar to the basic *direct_experience* function. The only difference is that, as shown in line 14, the evaluation history entries are filtered using the identifier of an agent, instead of the one of a resource. This means, that *get_combined_direct_experience_for_agent* calculates a trust value for any given agent by calculating the average of the final trust values created by past evaluations of resources provided by this agent.

```
1  def direct_experience(agent, resource_id, recency_limit, scale, ↙
       ↳ logger):
2      history_lines = logger.read_lines_from_agent_history(agent)
3      # getting all history values of the agent respective to the ↙
       ↳ evaluated resource and filters them based on their age
4      # and the recency limit set in the trust preferences of the ↙
       ↳ agent
5      history = [float(entry['trust_value']) for entry in ↙
       ↳ history_lines if entry['resource_id'] == resource_id and
6              datetime.strptime(entry['date_time'], ↙
       ↳ BasicLogger.get_time_format_string()) > recency_limit and
7              entry['trust_value'] != 'None']
8      # calculate direct experience
9      direct_xp = sum(history) / len(history) if len(history) > 0 ↙
       ↳ else None
10     return direct_xp
11
12 def get_combined_direct_experience_for_agent(agent, third_agent, ↙
       ↳ logger, recency_limit, scale):
13     history_lines = logger.read_lines_from_agent_history(agent)
14     history = [float(entry['trust_value']) for entry in ↙
       ↳ history_lines if entry['other_agent'] == third_agent and
15             datetime.strptime(entry['date_time'], ↙
       ↳ BasicLogger.get_time_format_string()) > recency_limit and
16             entry['trust_value'] != 'None']
17     # calculate direct experience
18     direct_xp = sum(history) / len(history) if len(history) > 0 ↙
       ↳ else None
19     return direct_xp
```

List of Listings 4.4: Calculation of the *Direct Experience* value and functionally related helper function

The final example given in listing 4.5 introduces communication between agents for trust evaluation purposes. The calculation begins by creating a list of third-party agents – agents that are neither the truster or trustee – which is shown in line 2. The same line also implements a filtering mechanism to avoid contacting agents whose average evaluation result (see listing 4.4) falls below the cooperation threshold of the current agent. This step helps preventing agents, who are not considered trustworthy enough for an interaction, from polluting the *Recommendation* value with possibly malicious testimony. In the third line, the sub-function *ask_for_recommendations* is called to handle the actual communication with other agents. Finally, the median of the received testimony trust values is calculated. The median was chosen for this calculation in particular because this method for calculating the average of the received recommendations is less prone to statistical outliers than the arithmetic mean. However, an average value can only be calculated for sets that are not empty, so in case that no witness testimony was received, the *Recommendation* value will be undefined.

ask_for_recommendations works by first building the request message to send to the other agents, which are given as function parameter *agents_to_ask*. This message consists of three parts: (1) a string signaling which factor is being used and thereby implying the format of the expected response, which differs based on the evaluated factor, (2) the identifier for the currently evaluated resource, and (3) the lifetime limit given by the requesting agent's *Recency* settings to allow response customization. After crafting the message, every agent given by the caller is contacted using the function *ask_other_agent* in line 11, which is an abstraction of the low-level network transmissions provided by aTLAS. If the received value is not undefined – which is tested in line 12 – it is weighted based on the average evaluation result for the third-party agent and added to the list of witness testimony values, which is returned to the caller function after contacting all agents.

The aforementioned response value is calculated the same way as the *Direct Experience* value for the given resource, as shown in line 20. The surrounding function is called by the aTLAS networking code, which then sends the value back to the requester via the opened network connection.

```
1 def recommendation(agent, other_agent, resource_id, scale, logger, ↙
    ↳ discovery, recency_limit):
2     agents_to_ask = [third_agent for third_agent in discovery if ↙
    ↳ third_agent != agent and third_agent != other_agent and ↙
    ↳ get_combined_direct_experience_for_agent(agent, third_agent, ↙
    ↳ logger, recency_limit, scale) >= ↙
    ↳ scale.minimum_to_trust_others()]
3     recommendations = ask_for_recommendations(agent, resource_id, ↙
    ↳ agents_to_ask, scale, logger, discovery, recency_limit)
4     return statistics.median(recommendations) if ↙
    ↳ len(recommendations) > 0 else None
```

```
5
6  def ask_for_recommendations(agent, resource_id, agents_to_ask, ↙
       ↳ scale, logger, discovery, recency_limit):
7      recommendations = []
8      message = ↙
       ↳ f"recommendation_{resource_id}_{datetime.strftime(recency_limit, ↙
       ↳ BasicLogger.get_time_format_string())}"
9      for third_agent in agents_to_ask:
10         remote_ip, remote_port = discovery[third_agent].split(":")
11         response = ask_other_agent(remote_ip, int(remote_port), ↙
       ↳ message)
12         if response != 'None':
13             received_value = float(response)
14             recommendations.append(
15                 get_combined_direct_experience_for_agent(agent, ↙
       ↳ third_agent, logger, recency_limit, scale) * received_value
16             )
17     return recommendations
18
19 def recommendation_response(agent, resource_id, recency_limit, ↙
       ↳ scale, logger):
20     return direct_experience(agent, resource_id, recency_limit, ↙
       ↳ scale, logger)
```

List of Listings 4.5: Calculation of the *Recommendation* value

This concludes the explanation of the created implementation of CRAB. In the next chapter, this implementation will be tested towards its feasibility and fulfillment of the requirements created in chapter 2.1, its scalability regarding large amounts of agents and exchanged messages, and its *Accuracy* in calculating the trustworthiness of resources.

5 Evaluation

The concept described in chapter 3 and the resulting prototype presented in chapter 4 were developed with the requirements arising from the problem description (see section 1.3) in mind. Evaluating CRAB's implementation in regard to these requirements is part of the feasibility analysis conducted in section 5.1. It includes verifying the fulfillment of the requirements which were established in section 2.1 along with a thorough analysis of the runtime complexity of the trust evaluation process. After examining whether the given implementation adheres to the prerequisites posed by the problem, a scalability analysis is performed in section 5.2 to examine, how scaling up the number of web applications and exchanged messages affects the average runtime of the trust evaluation process. This can be understood as practical extension of the theoretically applied requirement R1, as it makes the same assumptions about the importance of a short runtime for trust evaluations (see section 2.1.1). However, using only the requirements or scalability as metrics for the evaluation of the concept and implementation of CRAB, its precision for evaluating trust cannot be judged. For this reason, the *Accuracy* of the computed trust values will be tested in section 5.3.

5.1 Feasibility Analysis

In section 2.1, 5 requirements R1-R5 were described to rate the quality of trust evaluation models. The resulting score for each requirement is listed in table 5.1. To compare the trust model presented in this thesis with the groups of entity trust-based trust models that have been described and analyzed in chapter 2, the analysis results for those groups of trust models have been included in the table as well. Before that, however, each requirement will be discussed individually in order to explain the results for the analysis of the CRAB trust model.

5.1.1 R1 - Trust Evaluation time

Evaluating trust for every potential interaction creates overhead for solving problems with the help of other network participants like service or information providers. As this takes time and consumes computational resources that cannot be applied for

solving the problem the application is used for, this is clearly undesirable. This requirement is not evaluated on a scale, but instead presented as is, with lower values being better than larger ones. The runtime of an algorithm strongly depends on the specific use case, as well as hardware, network, and environmental conditions. Thus, the runtime complexity of the algorithm expressed in big O notation is used as metric for this requirement instead (see section 2.1.1). However, practical runtime experiments are be presented in section 5.2 to provide some context for the trust evaluation process.

The runtime complexity for the implementation is calculated in terms of the number of web applications n that are stored in the Discovery of the truster web application and the number of messages m that have been exchanged between these applications and the truster application so far. To simplify the calculation, it is assumed that every content trust factor implemented in chapter 4 is also used in the calculation. In the trust evaluation function, all content trust factor values are calculated synchronously and in order. Before these values can be calculated however, initialization code is executed to prepare the necessary data structures and calculate the *Recency* limit. These instructions run with a runtime complexity of $O(1)$. Similarly, calculating the final trust value is also done in constant time and the repeated code block surrounding the call of the sub-functions for the content trust factors performs a range of dictionary and file operations, both of which are executed with $O(1)$ as well. However, some of the aforementioned sub-functions require access to the evaluation history and the peer group dictionary. The evaluation history in the current version of aTLAS is a list of trust evaluation entries which need to be filtered as needed for the current use-case. Therefore, looking up data in the evaluation history creates a worst-case complexity of $O(n)$. Similarly, enumerating the peer group dictionary – e.g., for contacting peers to ask for recommendations – can only be performed with a worst-case complexity of $O(m)$. As a result, calculating the *Topic* trust value creates a complexity of $O(m^2)$ and calculating *Recommendation* or *User Expertise* values requires a complexity of $O(m * n)$. Because m and n are mathematically independent from each other, the resulting runtime complexity of the trust evaluation function is case distinct. Thus, if $m \geq n$, the complexity of a single trust evaluation is $O(m^2)$. In other cases, the complexity equals $O(m * n)$.

As mentioned above, m describes the number of messages exchanged between the truster application and its potential interaction partners at the point of the trust evaluation. This value can therefore also be understood as the current number of entries in the evaluation history. Limiting the number of entries to a constant value, and thus making m a constant value, transforms the previous, approximately quadratic runtime complexity $O(m * n)$ into a linear runtime complexity equal to $O(n)$. In these cases, the runtime would depend only on the number of peer agents. However, this approach presents a new problem. The evaluation history is an important part

of CRAB's trust evaluation and limiting its number of entries negatively affects the quality of the trust value calculation within the trust model. Yet, a sufficiently large, well chosen constant value for m might significantly enhance the performance of the model with a tolerable negative effect on the trust evaluation quality.

5.1.2 R2 - Decentralization

Using a trust model with a centralized structure for evaluating trust in a decentralized web obviously defeats the purpose of decentralization and should therefore be avoided (see section 2.1.2). Consequently, CRAB was designed and implemented to work without centralized authorities or agents with elevated or administrative powers. Each agent is evaluating the trust for its interactions locally and only communicates with its equal peers to gather recommendations. This means, CRAB meets the highest standard for R2, as defined in section 2.1.

It should be noted that this assessment only applies to the model itself, as aTLAS uses an architecture with centralized structures to simulate trust scenarios (see [4]). However, aTLAS is only the testbed used for individually testing the trust model, and not part of the CRAB model concept.

5.1.3 R3 - Need For Human Action

It is not feasible for service providers in a decentralized web to employ humans to evaluate trust manually because of the massive amount of potential interaction partners [4]. Based on this information, the intention behind requirement R3 is to lower the amount of financial overhead generated by requiring humans to be involved in the trust evaluation (see section 2.1.3).

CRAB does not require user input during the trust evaluation. However, as the model makes heavy use of the trust preferences, the respective values have to be configured before the model can be used as intended. Further, configuring the preferences requires expert knowledge to properly define the relevant thresholds, context levels and factor specific values. This represents a usability disadvantage of the CRAB model compared to other trust models that do not require any or professional configuration. Even if the configuration process is automated with appropriate default settings for the agents's given situation, the trust preferences might still need to be fine-tuned to reach the model's full potential. Therefore, CRAB can only achieve the second highest score for requirement R3.

5.1.4 R4 - Situational Awareness

Trust in the decentralized web is dynamic [4] and therefore highly situational. This means, that, to properly assess trust, a trust model needs to analyze the situation in which the planned interaction is supposed to happen (see section 2.1.4). CRAB's content trust concept as explained in chapter 3 uses a plethora of different contentual information as well as contextual metadata to calculate trust values. This promotes the dynamic adaptation of the model's trust evaluation to various conditions. Thus, CRAB fulfills all aspects of R4 and therefore receives the highest possible score for this requirement.

The situational awareness of a trust model matters because it helps to create trust in situations where e.g. trusted peer applications have been exploited based on a vulnerability present in their software and are now used to relay malicious resources. An entity-trust based trust model would require bad experiences to happen to the truster agent before the respective applications are not trusted anymore. A dynamic content trust model like CRAB however, would be able to assess the resource individually and prevent bad experiences for the truster by calculating a low trust value. After the original owners regained access to the device and the peer agents are providing high-quality information again, these resources would be awarded high trust values and the overall information quality would not suffer noticeably for the truster agent during and after this time. This is contrasted by entity-trust based trust models, as they are more inert and a once hacked peer might be excluded from future interactions based on low trust values in past.

5.1.5 R5 - Trust Scale

A well-crafted trust scale should empower the trust evaluation to create trust values that capture some of the dynamics of trust and, ideally, allow for an arbitrarily precise trust value that makes comparing potential interaction partners easier and more productive (see section 2.1.5). CRAB achieves this by providing a generic interface that allows any float or integer-based trust scale to be defined in the trust preferences of a web application. This means that every web application can enforce its trust scale preferences freely and adapt it to the current situation, all while maintaining precise trust values. Therefore, all aspects for requirement R5 are met.

5.1.6 Comparison with the Previously Analyzed Trust Model Groups

Finally, the trust model groups defined in section 2.2 are compared to CRAB in table 5.1. Due to space limitations, the names of the objects of comparison are abbreviated in the evaluation matrix, but are given in the following key:

Dir = Direct Trust-based trust models
Rep = Reputation-based trust models
SC = Socio-Cognitive trust models
Org = Organizational trust models
CRAB = The trust model presented in this thesis

	Requirements				
	R1	**R2**	**R3**	**R4**	**R5**
Dir	$O(n)$	★★★	★★★☆ to ★★★★★	★☆☆☆	★★★
Rep	$O(n)$ to $> O(n^2)$	★★★	★★★★	★★☆☆	★★★
SC	-	★☆☆ to ★★☆	★★☆☆ to ★★★★☆	★★★★☆	★★★
Org	-	★☆☆ to ★★★	★★★★	★☆☆☆	★★☆
CRAB	$\begin{cases} O(m^2) & m \geq n \\ O(m * n) & \text{otherwise} \end{cases}$	★★★	★★★☆	★★★★★	★★★

Table 5.1: Comparison between the previously analyzed groups of trust model design approaches and CRAB in regard to requirements fulfillment

As the comparison matrix in table 5.1 shows, CRAB performs well in regard to most requirements. A noticeable disadvantage is the need for the manual configuration of trust preferences, which most of the previously analyzed trust models do not require. Further, the runtime complexity of the developed trust model exceeds the complexity of both Direct Experience-based trust models and Reputation-based trust models. However, since the majority of the papers analyzed in chapter 2 do not provide any information about the runtime complexity of their presented models, this comparison is only representative to a limited extent. CRAB, on the other hand, shows its strengths especially in terms of situational awareness as defined in requirement R4, and exceeds every analyzed group of models in this regard.

5.2 Scalability Analysis

In this section, the scalability of the created implementation is examined. To test a range of varying combinations of agents and messages exchanged between them, 3 test runs are prepared. These runs are used to observe the effects of different size scaling on the trust evaluation runtime. Each test run uses every implemented content trust factor and consists of different scenarios that are randomly generated based on fixed agent and message size constraints. The sizes for each scenario ordered by test run are given in tables 5.2, 5.3, and 5.4.

Scenario Size	**# of agents**	**# of exchanged messages**
5	10	5
25	10	25
50	10	50
75	10	75
100	10	100
250	10	250
500	10	500

Table 5.2: Scenarios for Test Run A

Scenario Size	**# of agents**	**# of exchanged messages**
5	5	10
25	25	10
50	50	10
75	75	10
100	100	10
250	250	10
500	500	10

Table 5.3: Scenarios for Test Run B

All measurements were taken on the same system using an Intel® Core™ i7 8700 in combination with 32 Gigabytes of memory. Both, the aTLAS web server, as

Scenario Size	# of agents	# of exchanged messages
5	5	5
25	25	25
50	50	50
75	75	75
100	100	100
250	100	250
500	100	500

Table 5.4: Scenarios for Test Run C

well as the supervisor with enough capacity for the agents contained in the current scenario are run on this system simultaneously. To measure the runtime of each trust evaluation, an aTLAS built-in function is used to record the time elapsed during the current trust evaluation and writing it to a file, in combination with the index of this evaluation. Finally, the total preparation, execution, and cleanup time for the whole scenario is printed to the console. To mitigate the statistical effect of fluctuations based on processor scheduling or CPU load, each scenario is run 3 times and the average of these results is calculated.

In figure 5.2, the average execution time for the trust evaluations performed for the given scenarios for all 3 test runs is displayed. The average execution time is created by measuring the execution time for each trust evaluation in the current scenario and then calculating the arithmetic mean of those values. The difference between individual evaluation execution times can vary greatly, as trust evaluations might be canceled prematurely because an attempted deception was noticed or the maximum resource lifetime exceeded, while other trust evaluations run to completion. It should be noted that the y-axis is scaling logarithmically because the average execution time per trust evaluation varies strongly between the 3 test runs. As the figure shows, the average execution time grew slowly and not monotonously during test run A. The difference in execution time between the trust evaluations in the different scenarios of this run is low. As a result, the average execution time per trust evaluation scaled by a factor of 2.57 from scenario size 5 to 500, while the number of messages scaled by a factor of 100. In stark contrast to this, test run B leads to a big leap in average execution time. It starts out below run A, but scales past it after the first scenario, up to an average execution time per trust evaluation of approximately 43 seconds. This means the average execution time scaled by a factor of 629.27 from

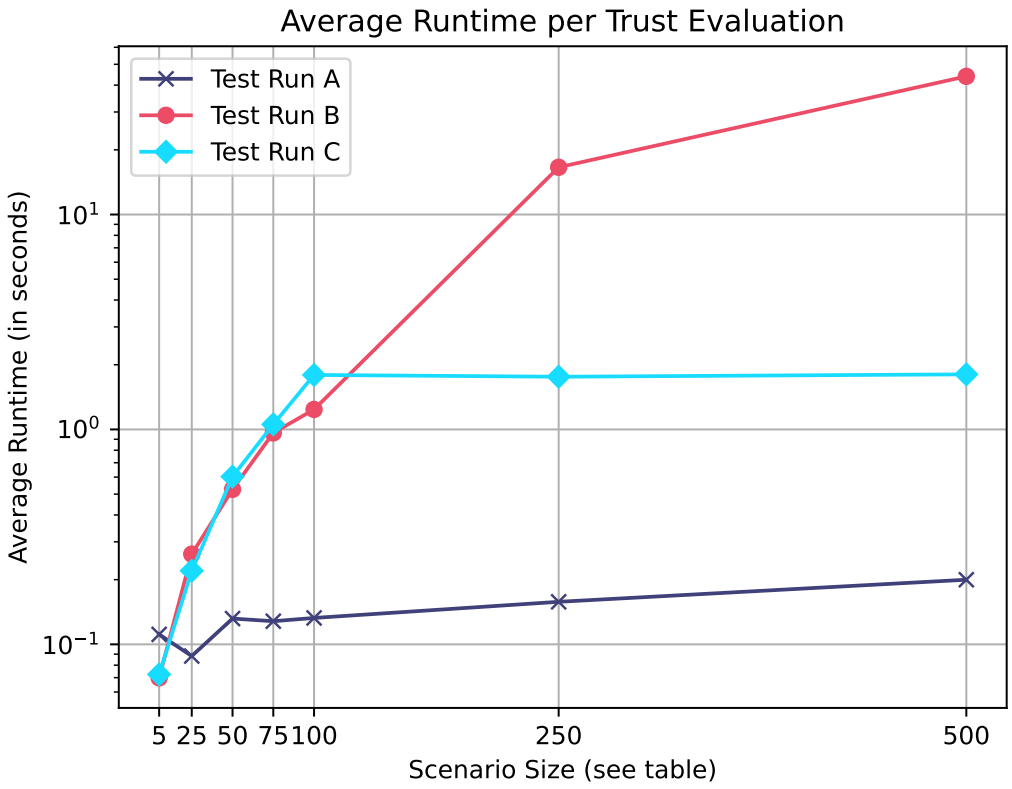

Figure 5.1: Average execution time per trust evaluation for different scenario sizes and configurations

scenario size 5 to 500, while the number of agents scaled by a factor of 100. The reason for this conspicuous pattern lies in the structure of the aTLAS testbed, as it simulates each agent in the scenario with a custom thread and communication socket. This overloads the used test system and results in abnormally high average execution times. This phenomenon is exclusive to the testbed and does not affect potential real-world implementations because in those situations, each agent only has to execute its own trust evaluation process. Finally, test run C places itself between the two previous ones, scaling similarly to test run B until scenario size 100 and then staying almost constant. This means the average execution time grew by a factor of 24.88 from scenario size 5 to 500, while the number of messages grew by 100 and the number of agents scaled by 20. This similar behavior shows, that the number of agents in a scenario has a more significant impact on the average execution time of trust evaluations than the number of messages. This seemingly contradicts the runtime complexity calculated in section 5.1.1, which presents the

number of exchanged messages equally influential to the runtime as the number of agents. But, the complexity is a theoretical term that does not consider the real runtime of instructions. Using more agents requires the testbed to open more network connections, perform more time-expensive network overhead (e.g., opening connections, sending control messages, etc.), open new threads for each agent, et cetera. This results in the actual runtime cost for adding a single agent to a scenario being significantly larger than adding another message, even if both numbers scale similarly. Thus, even if the number of messages matters for the runtime complexity of the trust evaluation, their practical impact on the runtime of a trust evaluation is negligible, especially in scenarios with many agents.

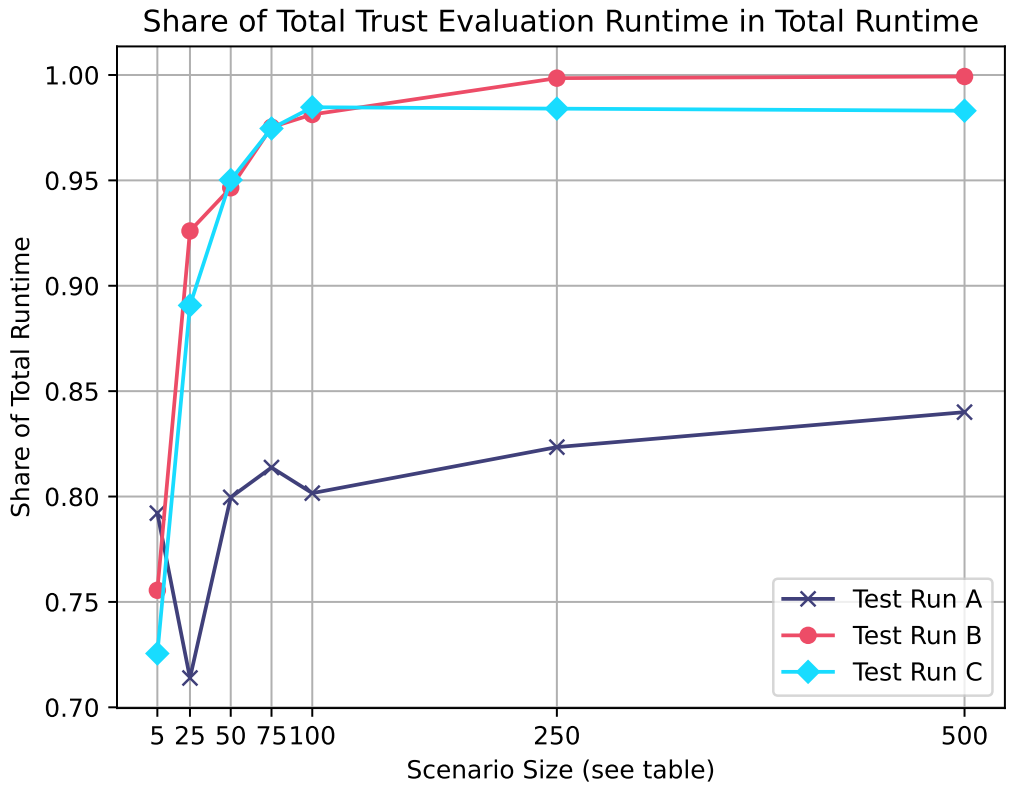

Figure 5.2: Share of total runtime that is required for the performed trust evaluations for different scenario sizes and configurations

To provide some context on the trust evaluation process simulated by aTLAS, the share of total runtime that is used for trust evaluations is presented in figure 5.2, again, ordered by test run. This value is formed by calculating the sum of execution times for the trust evaluations and dividing it by the total execution time for the

current scenario. It should be noted that this total execution time does not contain the runtime that is required for initializing or cleaning up the testbed. It only contains the time required for actually simulating the scenario.

What stands out on this graph is, that both test runs B and C scale and behave in a similar way: starting out at a low proportion of the total execution time and quickly rising to almost 1. However, the recognizable pattern is similar to figure 5.2, meaning that C scales monotonously, similar to B, until scenario size 100. From this point on, the share of execution time used for trust evaluations stays constant during test run C, while it grew further in test run B. This is similar to the average execution time per trust evaluation as described above. Noticeable ties between both figures also show for test run A, with its share of execution time growing slowly and not monotonously, while not reaching up to test runs B or C by far.

5.3 Accuracy Analysis

While the two previous evaluations examined the CRAB implementation in regard to its software functionality and fulfillment of the posed requirements, the *Accuracy* analysis will examine, how precise the created solution – CRAB – is at calculating computational trust. The already mentioned metric that will be used for this analysis is *Accuracy* as described by Jelenc et al. [28]. This metric can be used to evaluate trust models that do not have a decision making mechanism, which is the case for CRAB. In short, *Accuracy* expresses the similarity between the ranking of agents by trust value as calculated by the evaluated trust model, and the ranking of agents by capability. The capability of an agent describes its quality as an interaction partner, which is precisely the value that the trust evaluation process attempts to estimate. For a more in-depth explanation for *Accuracy*, as well as the calculation method and definition, see [28].

Using an already established metric allows for a comparison of the CRAB model with different existing trust models based on the results presented by Jelenc et al. [28] for the respective models. Because the authors calculate *Accuracy* using their Alpha testbed, which works differently to the aTLAS testbed used in this thesis, some differences have to be mitigated by means of careful scenario design. The main difference between the two testbeds is, that the Alpha testbed uses only one truster agent α in each scenario to calculate trust values and make trust-aware decisions. aTLAS, however, allows an arbitrary amount of truster agents. This is mitigated by using aTLAS scenarios with exactly one truster agent, called A.

In addition to A, 50 trustee agents with randomly generated capabilities are added to each scenario. All agents in the scenario are fully connected to each other and have

an opinion about other agents, in accordance with the evaluation scenario designed by Jelenc et al. To simulate each agent providing the same service, which is posed by the authors as another requirement, the message itself, as well as the topics of the message are similar between all observations exchanged in the scenario.

While the Alpha testbed generates opinions on the fly, the opinions used in this evaluation are based on history data specified in the aTLAS scenario file. To generate history data that is based on the capability of an agent, but contains sufficient noise to be realistic, a pseudo-random generator based on a truncated normal distribution is used, similar to the one, Jelenc et al. used for modeling interactions. This normal distribution is parameterized with the capability of an agent and an arbitrary standard deviation. The size of the standard deviation is responsible for the range of the resulting noise in the generated data. The standard deviation used for the entirety of the evaluation is $\sigma = 0.10$, similar to the one applied by Jelenc et al. The authors use a different, smaller deviation ($\sigma = 0.05$) for generating opinions with less noise than interaction outcomes. They do this to simulate the fact that opinions are in reality often closer to the capability of an agent than the outcome of a single interaction. For simplicity, in this experiment, this effect is achieved by aggregating multiple values with the regular amount of noise and calculating the respective opinions as the median of previous and historic interactions, as described in section 3.2 for the *Recommendation* content trust factor. Similar to the scenario history, all content trust factor values (e.g., *Bias*, *Specificity*, *Likelihood*, etc.) provided in the observation object are randomly generated with the mentioned generator.

The messages in a scenario are sent in cycles. During each message cycle, every trustee agent in the scenario sends exactly one message to A, which then computes the trust value for this message and writes it to the evaluation history. This data is extracted and the *Accuracy* for the current cycle calculated as described in [28] based on the agent capabilities. Contrary to the models implemented in the Alpha testbed, CRAB calculates the trust value for each resource. But as each agent sends its own resource with exactly one message, this trust value can be inferred to the agent itself and used for the ranking required for the *Accuracy* calculation.

CRAB is heavily customizable and as such may perform differently in varying situations. To test this, 4 different scenarios have been created, each testing a different situation with a respective configuration. For all of these scenarios, the content trust factor *Deception* is either deactivated or the *Deception* threshold is set to -1.0 to prevent the trust evaluation from ending prematurely with a trust value of -1.0, as this would strongly lower the *Accuracy* for agents with low capabilities. This is not an issue in real-world scenarios, as it would dissuade the truster agent from interacting with deceptive resources. The exact trust value does not matter in these cases. The *Deception* value is still included as a data point for each message. The *Accuracy*

evaluation scenarios follow the basic principles outlined above, but differ from each other in the aspects presented below:

1. **Scenario Acc1:** Agent A uses every implemented content trust factor for its trust evaluation, but there are no authorities, trusted authors or trusted topics configured. Each service provider specifies only its own name as author and does not present any references to related resources. This simulates a situation in which the agent does not know its peers and the network does not contain any authorities known to agent A.

2. **Scenario Acc2:** In this scenario, only the content trust factors *Direct Experience* and *Recommendation* are used by A. As aTLAS currently does not simulate trust-aware decisions, an interaction experience value similar to the one calculated by the Alpha testbed is not available. For this reason, each message contains a *Specificity* value, which is supposed to give the model per message input, similar to the effect of an interaction experience value. This scenario is included for reference, to test how including more content trust factors affects the resulting *Accuracy*.

3. **Scenario Acc3:** In this scenario, similar to scenario Acc1, each content trust factor is in use. In the configuration of agent A, 5 randomly selected agents are assigned as authorities and the 5 agents with the highest internal capabilities are set as trusted authors. Finally, the generator based on the parameterized truncated normal distribution is used to assign A's *Topic* trust values for each trustee agent in the scenario. This scenario simulates an advanced and more dynamic situation with possibly untrustworthy authorities, already formed *Topic* trust values and well chosen trusted authors.

4. **Scenario Acc4:** This scenario is mostly equivalent to Acc3, with the only difference being that no authorities are chosen. Having no artificially created hurdles, this scenario should be considered the main benchmark for evaluating CRAB's *Accuracy*.

The results of the *Accuracy* evaluation are displayed in figure 5.3. Since the graphs for the results are relatively close to each other in this chart, which makes it hard to clearly differentiate between them, figure 5.4 shows a zoomed in view of the same data on a more granular y-scale. In addition to the graphical representation of the *Accuracy* scores over time, figure 5.5 illustrates the results of the experiment as boxplot. All three figures are based on the same dataset.

CRAB evaluates trust with high *Accuracy*, which is proven by the results of the evaluation experiment. Looking at figure 5.5, it is evident that Acc4 is the scenario for which the model achieved the highest *Accuracy*. CRAB showed the lowest *Accuracy*

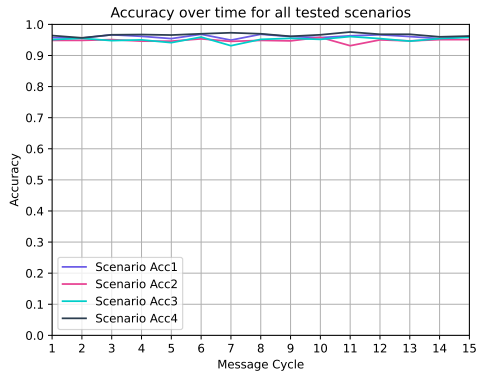

Figure 5.3: *Accuracy* over time for all tested scenarios

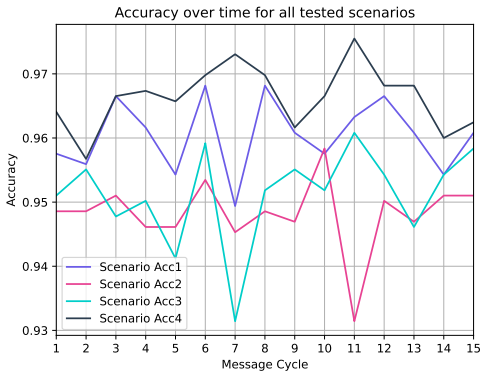

Figure 5.4: In closeup view for the graphs shown in figure 5.3

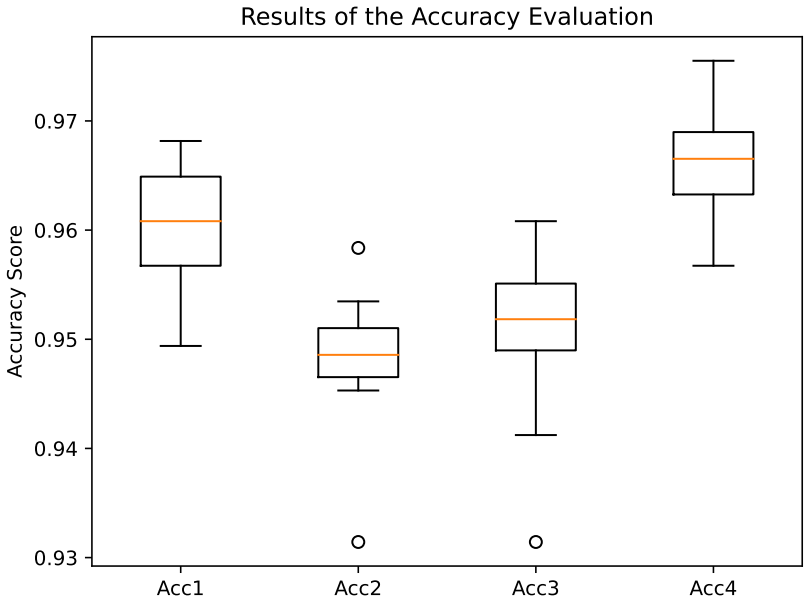

Figure 5.5: Overview of the results of the *Accuracy* evaluation

for scenario Acc2. Based on the results of the trust models examined by Jelenc et al. [28], CRAB is, under the circumstances and assumptions of the conducted experiments, more accurate than the trust models proposed by Yu, Singh and Sycara [29], and Abdul-Rahman and Hailes [30], as well as EigenTrust [31], Travos [32] and BRS [33]. This enumeration contains all trust models examined in [28].

6 Conclusion

The current state of the web is in many aspects characterized by privacy and accessibility concerns with a user hostile power dynamic being exerted from centralized companies and authorities. One way to combat this process is to push for a re-decentralization of the web. However, this creates new problems deriving from the large amount of unknown and potentially malicious information providers that exist in a decentralized web. Instead of being able to manually choose from a handful of central information providers, users will be forced to turn to computational trust to make trust-aware decisions in their stead. Computational trust is by no means a new topic. Thus, respective scientific work, especially in the form of trust models and trust management systems, already exist. But, these solutions are often tailored towards a centralized web architecture and evaluate trust on an entity basis, instead of evaluating for each message individually. This does not suffice for a decentralized web since it is a more dynamic environment. A possible solution for this is the use of content trust, which works on top of entity trust and evaluates the trustworthiness of each message individually and takes a wide range of contentual and contextual data into consideration to perform more in-depth trust evaluations. However, while the concept of content trust already exists, it is not yet clear, which content trust factors can or should be implemented to work for a decentralized web with autonomous web applications. Designing and building these implementations is the main objective of this thesis.

To analyze the current state of the art in computational trust models, a set of requirements for evaluating these solutions was defined and the existing trust models were divided into groups of approaches based on their information source. These groups were presented and explained based on the chosen examples and later evaluated. The results of the evaluation of these groups show, that the previous solutions fulfill most requirements well already, but do not possess situational awareness. Thus, the existing knowledge and technology in regard to computational trust models can be used as the basis for a new trust model that, in addition to existing trust information sources, uses content trust factors to achieve more advanced situational awareness. Therefore, to prepare for designing the concept of the trust model, four dimensions of trust were extracted from the analyzed literature.

Building on top of the trust dimensions derived in the previous chapter, an implementation concept called CRAB (**C**ontent t**R**ust ev**A**luation model for the distri**B**uted

web) was designed and presented. In this concept, the trust evaluation is part of a bigger Trust Awareness subsystem. This system provides an abstract interface for the in- and output of the trust evaluation. The correct analysis and classification of input data is assumed. Based on this conceptual design, the model was implemented for the aTLAS trust testbed. In chapter 4, this implementation was presented in excerpts to explain the inner workings of the trust evaluation process.

Finally, the implementation was evaluated in three parts. In the feasibility analysis, the model fulfilled all defined requirements perfectly and performed better than the previously analyzed models, except for R3 - *Need for Human Action*. One point was deducted for this requirement, since the model's use requires manual configuration. Further, CRAB's runtime complexity is relatively high, but this requirement could not be compared well, as most of the analyzed papers did not provide sufficient information to calculate the runtime complexity of their presented models. The *Accuracy* analysis showed, that the trust evaluation precision of the CRAB trust model is very high and exceeds the *Accuracy* of every model tested by Jelenc et al. in [28]. It further showed, that using most of the content trust factors provided an advantage over just using *Direct Experience* and *Recommendation*. This advantage, however, was relatively small in the shown experiments. The model also demonstrated good stability against potentially malicious authorities. The third analysis was used to evaluate the scalability of the implementation in regard to the number of agents and messages in a scenario. The experiments showed, that the average execution time per trust evaluation scales well with a growing number of messages. However, the execution time does not scale well with a growing number of agents. This is, to some degree, caused by the used test setup, which simulates each agent on the same system, which negatively affects the performance of a single agent in large scenarios.

A possible improvement for lowering the time complexity of the current CRAB implementation and accelerating the trust evaluation would be to use a high performance database management system for managing the evaluation history data. Currently, this data is kept in a list which has a linear time complexity for looking up elements in the data set. A well optimized database or a more search optimized data structure (e.g., binary search tree) would allow for this data to be delivered in logarithmic or even constant time. However, as these optimizations create added complexity, they were omitted from the current prototype for reasons of code legibility.

The requirements that were defined in chapter 2.1 would have been more functionally complete with a metric for evaluating the precision of the trust evaluation performed by a trust model, similar to the *Accuracy* used in the respective analysis. But, because most of the analyzed trust model papers do not provide this kind of data, and the ones that do are not directly comparable in most cases, this was not possible in the current scope of the thesis. To compensate for this, the aforementioned *Accuracy*

experiments were conducted and showed the model performed better than the ones presented by Jelenc et al. [28]. However, their paper was published in 2013 and does not contain any of the more modern models presented in chapter 2. Thus, an *Accuracy* comparison between CRAB and modern entity trust-based trust models is subject to future work.

CRAB in its current implementation represents an early prototype of a trust model concept. This means the software is not yet ready for use in real-world scenarios because essential components of the trust awareness subsystem are still missing. Namely, these are the input classification component, as well as the context, content, and behavioral analysis components. Furthermore, the current model also requires a mechanism for creating initial trust in addition to validation mechanisms that ensure the received data is correct and has not been tampered with. All these components, as well as more testing of the model implementation, performance enhancements and creating standalone model implementations outside of a trust testbed, are subject to future work. Furthermore, more work in the area of accurate trust-aware decision making is necessary to ensure that the calculated trust values are used properly. Despite all these limitations, the high degree of *Accuracy* that CRAB achieved in the presented experiments suggests that this trust model can help solve the problems highlighted in chapter 1.3 and make content trust available for future web applications in the distributed web.

Bibliography

[1] L.-D. Ibáñez, E. Simperl, F. Gandon, and H. Story, "Redecentralizing the Web with Distributed Ledgers," *IEEE Intelligent Systems*, vol. 32, pp. 92–95, 2017.

[2] A. V. Sambra, E. Mansour, S. Hawke, M. Zereba, N. Greco, A. Ghanem, D. Zagidulin, A. Aboulnaga, and T. Berners-Lee, "Solid: A platform for decentralized social applications based on linked data," MIT CSAIL & Qatar Computing Research Institute, Tech. Rep., 2016.

[3] H. Yu, Z. Shen, C. Leung, C. Miao, and V. R. Lesser, "A survey of multi-Agent trust management systems," *IEEE Access*, vol. 1, pp. 35–50, 2013.

[4] V. Siegert, M. Noura, and M. Gaedke, "aTLAS: a Testbed to Examine Trust for a Redecentralized Web," in *Proceedings of the 2020 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)*. IEEE, 2020, pp. 411–416.

[5] Y. Gil and D. Artz, "Towards content trust of web resources," *Journal of Web Semantics*, vol. 5, no. 4, pp. 227–239, 2007.

[6] J. Granatyr, V. Botelho, O. R. Lessing, E. E. Scalabrin, J.-P. Barthès, and F. Enembreck, "Trust and Reputation Models for Multiagent Systems," *ACM Comput. Surv.*, vol. 48, no. 2, pp. 1–42, 2015.

[7] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," Mozilla, Tech. Rep., 2018. [Online]. Available: https://rfc-editor.org/rfc/rfc8446.txt (Accessed 12-September-2021).

[8] P. M. Stephen and Briggs, "Examining Trust, Forgiveness and Regret as Computational Concepts," in *Computing with Social Trust*, J. Golbeck, Ed. Springer, 2009, pp. 9–43.

[9] J. Sabater and C. Sierra, "Review on computational trust and reputation models," *Artificial intelligence review*, vol. 24, pp. 33–60, 2005.

[10] D. Zhang, F. R. Yu, R. Yang, and L. Zhu, "Software-Defined Vehicular Networks With Trust Management: A Deep Reinforcement Learning Approach," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1—15, 2020.

[11] M. M. Arifeen, D. Bhakta, S. R. H. Remu, M. M. Islam, M. Mahmud, and M. S. Kaiser, "Hidden Markov Model Based Trust Management Model for Underwater Wireless Sensor Networks," in *Proceedings of the International Conference on Computing Advancements*. Association for Computing Machinery, 2020.

[12] S. Jiang, J. Zhang, and Y.-S. Ong, "A Multiagent Evolutionary Framework based on Trust for Multiobjective Optimization," in *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)*, 2012.

[13] I. Pinyol and J. Sabater-Mir, "Computational trust and reputation models for open multi-agent systems: A review," *Artificial Intelligence Review*, vol. 40, pp. 1–25, 2013.

[14] E. Alemneh, S.-M. Senouci, P. Brunet, and T. Tegegne, "A two-way trust management system for fog computing," *Future Generation Computer Systems*, vol. 106, pp. 206–220, 2020.

[15] K. S. Barber and J. Kim, "Soft Security: Isolating Unreliable Agents from Society," in *Workshop on Deception, Fraud and Trust in Agent Societies*, 2003, pp. 224–233.

[16] X. Chen, J. Ding, and Z. Lu, "A Decentralized Trust Management System for Intelligent Transportation Environments," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–14, 2020.

[17] Q. Zhai and X. Xie, "Trust Management Model Based on Intuitionistic Fuzzy Information in Cloud Environment," in *Proceedings of the 3rd International Conference on E-Business, Information Management and Computer Science*. Association for Computing Machinery, 2020, pp. 561–565.

[18] G. Rathee, A. Sharma, R. Kumar, F. Ahmad, and R. Iqbal, "A trust management scheme to secure mobile information centric networks," *Computer Communications*, vol. 151, pp. 66–75, 2020.

[19] M. A. Azad, S. Bag, F. Hao, and A. Shalaginov, "Decentralized Self-Enforcing Trust Management System for Social Internet of Things," *IEEE Internet of Things Journal*, vol. 7, pp. 2690–2703, 2020.

[20] C. Boudagdigue, A. Benslimane, A. Kobbane, and J. Liu, "Trust Management in Industrial Internet of Things," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 3667–3682, 2020.

[21] C.-M. Mathas, C. Vassilakis, and N. Kolokotronis, "A Trust Management System for the IoT domain," in *Proceedings of the 2020 IEEE World Congress on Services (SERVICES)*, 2020, pp. 183–188.

[22] H. Fang, J. Zhang, and N. M. Thalmann, "A Trust Model Stemmed from the Diffusion Theory for Opinion Evaluation," in *Proceedings of the 12th international conference on autonomous agents and multi-agent systems (AAMAS 2013)*, 2013, pp. 805–812.

[23] R. Hermoso, H. Billhardt, and S. Ossowski, "Role evolution in open multi-agent systems as an information source for trust," in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, 2010, pp. 217–224.

[24] G. Lu, J. Lu, S. Yao, and J. Yip, "A review on computational trust models for multi-agent systems," *The Open Information Science Journal*, vol. 2, pp. 18–25, 2009.

[25] Z. M. Aljazzaf, M. Perry, and M. A. Capretz, "Online trust: Definition and Principles," in *Proceedings of the Fifth International Multi-conference on Computing in the Global Information Technology*, 2010, pp. 163–168.

[26] T. Grandison and M. Sloman, "A survey of trust in internet applications," *IEEE Communications Surveys Tutorials*, vol. 3, pp. 2–16, 2000.

[27] A. Flemming, "Qualitätsmerkmale von Linked Data-veröffentlichenden Datenquellen," 2011. [Online]. Available: http://www.dbis.informatik.hu-berlin.de/fileadmin/research/papers/diploma_seminar_thesis/Diplomarbeit_Annika_Flemming.pdf (Accessed 19-October-2021).

[28] D. Jelenc, R. Hermoso, J. Sabater-Mir, and D. Trček, "Decision making matters: A better way to evaluate trust models," *Knowledge-Based Systems*, vol. 52, pp. 147–164, 2013.

[29] B. Yu, M. Singh, and K. Sycara, "Developing trust in large-scale peer-to-peer systems," in *2004 IEEE First Symposium on Multi-Agent Security and Survivability*. IEEE, 2004, pp. 1–10.

[30] A. Abdul-Rahman and S. Hailes, "Supporting trust in virtual communities," in *Proceedings of the 33rd Annual Hawaii International Conference on System Sciences*. IEEE Computer Society, 2000, pp. 9–pp.

[31] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina, "The eigentrust algorithm for reputation management in p2p networks," in *Proceedings of the 12th International Conference on World Wide Web*, ser. WWW '03. New York, NY, USA: Association for Computing Machinery, 2003, pp. 640–651.

[32] W. T. L. Teacy, J. Patel, N. R. Jennings, and M. Luck, "Travos: Trust and reputation in the context of inaccurate information sources," *Autonomous Agents and Multi-Agent Systems*, vol. 12, pp. 183–198, 2006.

[33] A. Jøsang and R. Ismail, "The beta reputation system," in *Proceedings of the 15th Bled Electronic Commerce Conference*, 2002.

TECHNISCHE UNIVERSITÄT CHEMNITZ

Studentenservice – Zentrales Prüfungsamt

Selbstständigkeitserklärung

Name: Kirchhoff Vorname: Arved geb. am: 22.10.2000 Matr.-Nr.: 529538	Bitte beachten: 1. Bitte binden Sie dieses Blatt am Ende Ihrer Arbeit ein.

Selbstständigkeitserklärung*

Ich erkläre gegenüber der Technischen Universität Chemnitz, dass ich die vorliegende Bachelorarbeit selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe.

Die vorliegende Arbeit ist frei von Plagiaten. Alle Ausführungen, die wörtlich oder inhaltlich aus anderen Schriften entnommen sind, habe ich als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch nicht als Prüfungsleistung eingereicht und ist auch noch nicht veröffentlicht.

Datum: 18.01.2022 Unterschrift: .....

---

* Statement of Authorship

I hereby certify to the Technische Universität Chemnitz that this thesis is all my own work and uses no external material other than that acknowledged in the text.

This work contains no plagiarism and all sentences or passages directly quoted from other people's work or including content derived from such work have been specifically credited to the authors and sources.

This paper has neither been submitted in the same or a similar form to any other examiner nor for the award of any other degree, nor has it previously been published.