1 Introduction

Trust is the bedrock of society. From the evolution of species, global markets all the way to the modern sharing economy, trust has always had an important impact on almost every aspect of our lives. Trust is built on a good reputation which in turn is created through positive outcomes of past interactions. The value of a good reputation also depends on how widely known that reputation is. In small communities, knowledge about each other is gained through gossiping or personal experience. But local communities become less important as global communities and marketplaces become more common through internet based applications. Still, or even more so, those internet communities depend on trust. Protecting and distributing the knowledge about interactions in the digital world is a tough challenge we are faced with when designing a global trust system.

The current state-of-the-art trust building systems are online platforms for the sharing economy. The reputation systems of Uber[1] and AirBnB[2] are an essential part of their business. The good reputation allows commuters to trust their driver and get into the car of a stranger, or allows house owners to rent their home to a couple from the other side of the world. The reputation of drivers and renters are stored on the platforms, they are both valuable to the people as well as the company. This leads to problems when renters do not agree with updates to the platform: they cannot take their reputation and move to a competitor. Users are locked into those platform, giving platforms great power and influence.

A similar situation exists in the banking world. Banks are entrusted with their clients money, but their power led to corruption and the trust was abused. The situation escalated in the 2007-08 financial crisis which led to a global recession. The crisis inspired a new solution: Bitcoin. Bitcoin is supposed to enable secure payments without banks. As such it removes power from financial institutions and puts it back into the hands of the actual owners of the money.

Not every digital money transaction should require a bank, and similarly not every trustful interaction on the internet should require a third-party. Instead, the ability to prove one's trustworthiness on the internet should be open and free for anyone. Our vision is therefore to create a universal mechanism to create trust. This work sets an important step towards creating such a system. Specifically we propose a mechanism that protects and distributes the records of transaction which are essential for creating trust.

This first chapter introduces some key concepts around trust and explains the context of this work. It should shed light on the origin of trust research and its significance for the future of the internet. A thorough contextual basis is created for the reader to fathom the problem description and proposed solution in the following chapters.

1.1. Trust research

Virtually everyone that is part of a social community understands the concept of trust, yet defining trust scientifically is hard. This is also due to the fact that trust is studied in a diverse set of sciences: evolutionary biology, sociology, economics and lately computer science. In the simplified form of a model trust can well be described and studied. The prisoner's dilemma[6] is one such model from game

[1]https://uber.com
[2]https://airbnb.com

theory that creates a framework for understanding trust. It is widely used in research and is the basis for many experimental studies. We describe in the following the game, it's relation with cooperation and the impact on evolutionary theory, economics and computer networks.

1.1.1. Prisoner's dilemma

The game Prisoner's dilemma describes a dilemma common in many real-world situations, for example the problem of two partners caught for a crime that are questioned in two separate rooms. Each prisoner has two options, either deny all allegations, which is more generally called cooperating or betray the partner, which is called defecting in general game theory. If both stay silent, both will get a sentence of one year. If one betrays the other, the snitch is set free while the betrayed gets three years in prison. If both betray each other, they both have to serve two years. When analyzing the game without any additional knowledge and considering the payoffs for one of the prisoner's it is always advantageous to betray the other. Either the other also betrays, in which case two years is better than three, or the other stays silent in which case betraying sets us free. However, when considering both prisoners' outcomes together it would be best for both to stay silent.

While the game is quite simple the implications are far reaching. The game is able to show the connection between trust and cooperation. If both prisoners trust each other to never betray a partner, both will cooperate and get a small sentence, the best combined outcome. Yet any mistrust makes both fail at beating the system. The problem also describes many real world problems, called the tragedy of the commons. For example, the global warming is a problem that can only be solved if all peoples and all nations cooperate. Yet, the low cost and convenient usability of fossil fuels make it advantageous to defect and damage the environment. Either the others try to save the environment in which case a single defection will have a small impact, or the other will also damage the environment in which case a single cooperator will fail anyways. Only, if everyone trusts each other that everyone does the best they can to save the environment, then it is possible to beat the tragedy of the commons.

1.1.2. Evolution and cooperation

While cooperating, according to theory, is not necessarily a winning strategy, it is in our nature to do so, as has been shown by evolution theorists. The theory about competitive natural selection between individuals and mutation and inheritance of genes was the accepted truth about evolution since Darwin until in the late 1960s doubts arose about the completeness of this theory. When looking at group behavior in species one will find that cooperation is a common theme among related individuals, yet there is no place for cooperation in the classic Darwin theory [5]. In their work Axelrod and Hamilton [5] analyze how to combine the seemingly inferior individual's strategy of cooperating with the goal to maximize fitness. At the basis of their experiments is a again the Prisoner's Dilemma. Axelrod and Hamilton ran experiments on this game with multiple rounds instead of only one with different strategies. They found that if the game is played repeatedly with the possibility of meeting the same partner again in the future, cooperation between players can be established and be superior. Later research showed that this direct form of reciprocity, the act of returning a deed, is only one form of cooperation found in human behavior. Nowak and Martin [16] defined in total five forms in which cooperation can occur: kin selection, direct reciprocity, indirect reciprocity, network reciprocity and group reciprocity. Conceptually these forms can be described like this:

- kin selection: we help those that share our genes
- direct reciprocity: I help you, you help me
- indirect reciprocity: I help you, somebody helps me
- network reciprocity: neighbors help each other
- group selection: A group, in which members help each other, survives

Each concept entails at its basis trust. We trust our family, our group, our countrymen, those with whom we had a lot of shared experiences and those we heard good things about.

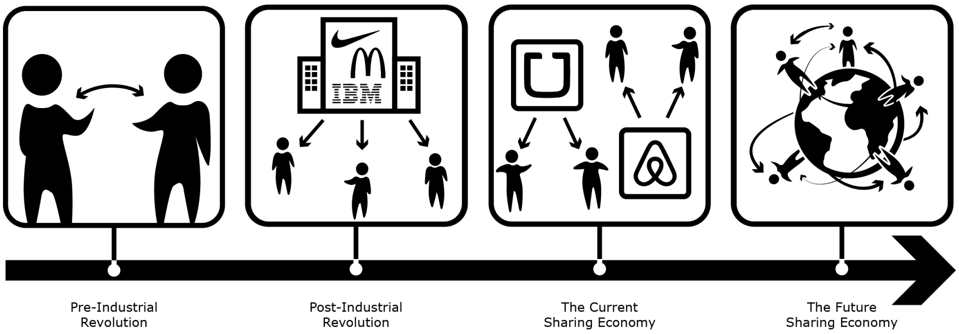


Figure 1.1: Evolution of the economy

1.1.3. Economics

Old: needs to be improved Trust also plays a major role in our economic system and has been for the the evolution of economy as shown in Figure 1.1. In the pre-industrial age, most economy and trade was done in local communities with families that trusted each other over generations and traders that returned year after year. During industrial and post-industrial age companies have largely replaced local producers and are trusted by millions of customers based on their brand name. Nowadays, in the information age, internet companies allow people to connect, trade and cooperate directly, examples being eBay for trading physical goods, AirBnb for sharing houses and Uber for ride-hailing. How trust and the lack of it influence trade and markets has been studied in the well-known paper by Akerlof [4]. He describes the information asymmetry between seller, who knows the quality of the goods which will be sold, and the buyer, who can only estimate that quality by some market statistic. The seller's incentive to sell goods of lesser quality than the average statistic leads to a decreasing statistic and thus price which in turn decreases the quality of the goods sellers are willing to offer for that lower price. Hence the market breaks down. Akerlof describes institutions to solve this problem namely institutions such as guarantees, brand names and certifications. These are trust inducing institutions and can be generalized as reputation systems. If the sellers sell goods to many people and those people report or gossip the good quality of what they have bought, others can trust those sellers and both seller and buyers will thrive. On the other hand a bad reputation will lead to a seller getting out of business as buyers will mistrust. This closes the gap to the work of Nowak as this reputation is what makes indirect reciprocity possible: the seller is not taking advantage of the buyer's inconvenient situation but the buyer cannot directly return that favor. Only by gossipping the event to other potential buyers who are then more willingly to buy from the seller is the reciprocity circle closed. [16]

1.2. Digital trust

Old: needs to be improved These analog, gossip-based reputation systems are what guide our decisions in buying cars, new or used, at which bank we store our money and at which restaurant we should have dinner. But reputation systems are also prevalent in the digital world: we make our decisions in buying used goods on eBay, renting a house to a stranger (or from a stranger), getting into a stranger's car (what mum told us not to) based on the reputation of the partner. The sharing economy or collaborative consumption is the rising star of economic concepts in the information age and it is power by reputation. A company offers a platform on which the two sides of a trade or transaction can find each other. With each encounter both parties can rate that interaction and it becomes part of their history. With a longer and more positive history the value of a profile increases as users see the reputation as security for a good interaction and are willing to pay for it. However, there are reasons for concern. What if the platform changes their rules in an almost unacceptable way or abuses the personal data their users have entrusted them with? Users cannot take their reputation and data to another platform because their reputation is actually owned by the platform facilitating the trades. Such abuses in which trusted companies act wrongly have been happening in many times in the past, examples being the dieselgate [1] and the facebook/cambridge analytica scandals. [2] These scandals show that even though millions of users trust them, central institutions do not necessarily serve their customers or clients.

1.3. Universal mechanism to create trust

Old: needs to be improved We envision a future in which collaborative consumption is possible without any intermediator. This future requires a reputation system which is application agnostic, owned by noone and ruled by everyone. A distributed reputation system as a layer directly on top of the internet. However this poses some challenges from a technical point of view. Distributed system are intrinsically hard to control and regulate, which is both blessing and curse. No party can impose unfair rules on other users but it is also hard to prevent malicious users from sending wrong information across the network. [10] reviews state-of-the-art reputation systems and finds that all commercial reputation systems are centralized. Some of the scientific reputation systems are decentralized like EigenTrust [12], P-Grid [3] and RateWeb [13], yet they have not been proven to work in settings where high throughput, global scaling are required which is the case for a global reputation system. Distributed, secure and globally scalable systems remain an unsolved problem.

1.4. TU Delft Blockchain Lab research

The ambition of creating the first global trust system is realized at the Blockchain Lab of TU Delft, which is the research group in which this thesis was created. The lab has a strong focus on exploring new concepts, implementing them and testing them in production grade software.

The research group has great experience and a solid track record in the field of distributed work systems. Especially peer-to-peer file sharing system have been studied, first and foremost the internally developed Tribler[3] application. Tribler is a client for the BitTorrent protocol. It offers many improvements over conventional BitTorrent clients like improved privacy and security, streaming and reputation management. It has been the testbed for algorithms of bachelor, master and Phd students for ten years with 1 million downloads in that period. In those years of research several milestones have been reached. In [14] we have solved the free-riding problem in the peer-to-peer file-sharing context with a reputation system that tracks uploads and downloads. With TrustChain [17] we have created our own blockchain fabric for bandwidth as a currency which builds on the previous work and adds tamper-proof recording and immutable history to the reputation system.

The problem of peer-to-peer file sharing systems maps well onto the trust domain. Users of BitTorrent clients download from other users who upload data. Downloading data has benefit to users because they are interested in the content, however uploading has no obvious advantage. It is only necessary to keep the content available for others. There is an obvious incentive problem, a tragedy of the commons. A free-rider can download without uploading, thereby consume resources without contributing. The problem can be solved through a trust system. By recording the behavior of each user and making it public a reputation can be assigned to each user which represents their resource usage. Users that contribute a lot increase their reputation while downloading decreases that reputation. Other users are able to inspect that past behavior of any potential partner and use it for their decision whether the partner deserves a contribution.

The recording of file transactions and security of those records is facilitated by our blockchain fabric TrustChain [17]. TrustChain is a multi-chain fabric which lets each user create an own chain. It is therefore built for horizontal scalability and unbounded throughput. By design, TrustChain enables the creation of trust in any application context. The solution is implemented in Tribler and has been used in production for more than a year as of 2018. The implementation details of TrustChain will be discussed more in Chapter 3.

The great scalability of TrustChain comes at the cost of security. The architecture allows for attacks to be detected but the system can only be secured if honest users engage in that defensive behavior. This can only be ensured through incentives: agents should never be able to gain an advantage by circumventing the rules.

1.5. Contribution

In this work we study a mechanism to ensure the proper dissemination and verification of transaction records. In any trust system, the transaction records are the basis for the reputation of users and thus the trust users have in each other. With the records, also the reputations itself are spread, leading to more agreement of reputation and thus higher value. Also defending the reputation records against

[3] https://tribler.org

attacks makes records more credible and dependable, opening applications for TrustChain with higher requirements for security.

We will enlarge on the problem description in the next chapter. Afterwards the problem will be defined formally and analyzed in the bounds of the definition. Before proposing a solution, some existing approaches for recording and dissemination of data will be discussed in chapter. Next, we define a solution based on the TU Delft blockchain fabric TrustChain and propose a specific mechanism of using such a fabric. Finally, we prove the correctness and scalability properties of the fabric in experimental analysis, before concluding and making suggestions for further research.

2

Problem description

Our audacious ambition is to create a global trust system. Such a trust system as a public, free and non-profit utility is a key element to enable the next-generation online applications. In the previous chapter we have introduced trust research and the influence on many real-life subjects. In this chapter we explain in more detail the problem at hand.

2.1. Model of trust and reputation

- Trust, Reputation and reciprocity have a circular relation ship
- Trust depends on Reputation

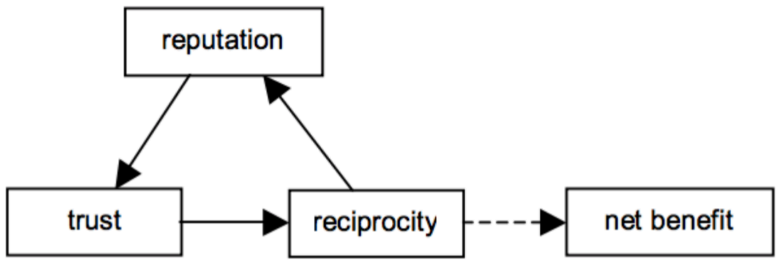


Figure 2.1: Circular relation between trust, reputation and reciprocity. Source: Mui, 2002 [15]

2.2. System architecture

As of today, no single algorithm or architecture can provide a solution that conforms with all these requirements. Only by combining multiple components and iterating their design can we approach a reputation layer that is able to conform with the requirements. This layer needs to combine these components:

Identity. At the lowest level there is the identity layer that ensures an entity is identifiable for other entities in the network. The most basic version of this is a simple public- private key pair for signing and encrypting data. But creating a new key pair is cheap, therefore this is not enough and in a later iteration of this identity system digital entities will need to be bound to real-world, verified entites like government-issued passports or biometric identifiers.

Communication. The internet creates a global communication network with high connectivity across the world but it is currently not in a state that direct communication between devices is straight-forward. The large increase in connected devices expended the address space of IPv4 and IPv6 transition has been slow, thus network address translation creates subnetworks with local adress spaces. Connection from such a subspace to a server with a public adress is still simple like it is the case with most client-server applcations on the internet, but direct communication when both devices are behind NATs is still not standardized. Also new routing solutions are required to ensure communication based on the actual identity layer mentioned before instead of the IPv4 and DNS identity layer.

Record and distribution mechanism Reputation is based on feedback on interactions. This feedback needs to be recorded and distributed, such that other entities in the network have a chance to respond to the history of feedback of their peers. Without a central entity which is aware of all transactions, each node will record some transactions. It is a challenge to create a global record which is correct, tamper-proof and well distributed across the network.

Interpretation of records Based on the recorded feedback each agent can interpret them to form an opinion about other nodes. For a reputation system the records are seen as positive or negative behavior and each agent can output a ranking of reputations for all peers this agent knows of. Those rankings are calculated based on a reputation function. In the past our research group has analyzed different reputation functions like NetFlow [17], MaxFlow [14] and PageRank [19].

Application layer We imagine that the reputation system we are developing can be used for any type of application that requires two entities to trust each other. This reputation layer will be accessible for anyone however no application will be able to delete data or lock data into their proprietary platform.

Each layer adds another level of protection: if identities are expensive and hard to create, fake identities will be less easy to create, protecting the system against spamming. Creating an immutable record and distributing that information to everyone makes knowledge tamper-proof, unchangeable and forever, making all information reliable. On the interpretation layer additional securities can be enabled: we envision a concept of locality to secure against distributed attacks with global collaborations of malicious nodes - if we only trust agents with a certain level of latency attackers can only choose from nodes in the vicinity and supply of those nodes is limited.

2.3. Value of reputation

The concept of trust is however vague. In the analog world, trust is more of a feeling than a rational calculation, yet computing systems are deterministic, precise and rational. The vagueness partly stems from the subjective interpretation of actions which in the digital system is the reputation function used, but also from the difference in what each person knows about another one, or in digitally speaking subsets of information that each one agent has. Yet, from a practical point of view it would be desirable if people could agree on the reputation and trustworthiness of agents, because it makes actions predictable and gives reputation its value.

To see this we should go back to the discussion of indirect reciprocity, which is one of the five forms of fostering cooperation which were introduced in chapter 1. People act pro-socially at a personal cost in order to build an altruistic reputation which is rewarded with third-parties acting pro-socially towards them. Reputation is valuable because people with higher reputation can expect more cooperation in future interactions. This indirect reciprocity mechanism works as long as agents agree on what is good and what is bad reputation. Once there is ambiguity about the reputation of agents this value decreases as even people with a bad reputation could be seen as good people by others due to that ambiguity. Also only if reputation has actual value to agents, we can ensure pro-social behavior in the network. Therefore we need agreement on the reputation of agents.

Agreement on the interpretation layer can be achieved by defining a function for all agents to use which calculates a quantitative reputation from the history of feedback. Usually this history is public, visible to everyone, however this is difficult to achieve in a distributed system. Here the second problem comes into play. The information a network node acts upon is a different subset of complete information on the network for each agent; each agent is in a different state. This situation is undesirable but inherent to systems with the requirements stated in the previous section. Thus, a first step towards agreeing on the reputation of agents is if agents agree on which data should be used as an input to the reputation function. In other words we have to make sure that agents disseminate their knowledge and obtain knowledge from other agents such that information is well distributed and available.

However, in most contexts sharing and obtaining information comes at a cost which is not negligible. Thus agents may be reluctant to spend resources without any direct reward. There is an obvious network effect to agents knowing more about their peers but agents can also gain reputation by co-operating with agents with low reputation. Thus there is no incentive to obtain a better view of the network.

2.4. Research question

The design of a global trust system bears many challenges. A lot of research has gone into the development of algorithms for calculating trust and models for the behavior of agents in the presence of trusted and distrusted agents. However, we found that the basis for trust is reputation, which in turn is based on the records of encounters. Yet, recording, distributing and securing the information about encounters in distributed systems is an underexplored topic and has only little literature published in the scientific community.

Consequently, the question that we are trying to answer is therefore:

How to record, disseminate and protect the records of interactions in a global trust system?

2.5. Related work

- First and second generation crypto currencies
- Reputation systems
- Dissemination mechanisms
- Conclusion nothing does it all

2.6. Agent behavior through incentive

2.7. Requirements of reputation systems

Reputation systems appear in many forms but we are concerned with their digital form as only digital networks can reach global scale with fast information distribution. Reputation systems have previously been formally defined to include at least three components [20]:

- Long-lived identities
- Recording and distribution of feedback about interactions
- Use of feedback to guide interactions

We need entities to be identifiable and in existence for a long time to ensure that future interactions between known entities are likely. If changing of identities is easy, a bad reputation is easily discarded and exchanged for a clean slate. We need to capture and distribute feedback of interacitons such that entities are aware of the history and reputation of other entities on the system. Finally, users should actually make use of the feedback on not just ignore it.

2.8. Requirements of global trust system

Next to the requirements of reputation systems we also have requirements for specific usecase of a global reputation system without centralized institutations.

- distributed: no entity should be owner of the reputation of all people, no single point of failure should exist
- scalable: future applications similar to those that exist today with centralized reputation systems should be able to handle billions of users.
- robust against strategic manipulation: once reputation increases in worth users of the system will try to exploit the system by attacking it, alone or by colluding and the architecture needs to be robust against such attacks

This introduces additional challenges: enforcing long-lived identities is even harder without the assumption of a trusted central entity that can check the validity of new entities. Also creating a central distributed record with synchronization at scale is a topic of ongoing research and generally seen as an unsolved problem. Finally, reputation system in general and distributed systems specifically are intrinsically weak in protection against malicious behavior, although they are robust in terms of complete failures as no single point of failure exists.

2.9. Scope

3

Formal Model Definition and implementation details

The discussion of distributed trust systems requires a well-defined framework. In this chapter we formally define a model in which we can analyze a new information dissemination mechanism. Also, we explain in detail how the previously introduced TrustChain architecture is implemented. This should guide the reader towards a proper understanding of the practical intricacies of our system.

We shall first reintroduce the notation defined by Mui in previous work on the subject of reputation systems. Afterwards we apply that notation to the TrustChain solution which will be the basis of the solution defined in later chapters of this thesis. Finally, we can specifically point out the shortcomings of that solutions. The next chapter will then introduce a new extension of TrustChain which will tackle those shortcomings.

3.1. Basic model and notation

For this model we use the notation that was defined by Mui in [15]. The goal is to develop the notation for a model of trust and reputation in a social network. Mui developed this notation to study a computation model for trust and reputation which fits the subject of this work perfectly. For the sake of simplicity we will further simplify the model and ignore the context dependence of the social networks, so the definitions assume the reputation and trust are about a single context.

We first define a social network in general.

Definition 1 (Social network)**.** A social network is a society of a set of agents $A = \{a_1, a_2, \dots a_N\}$ that allows for agents to communicate and interact with each other. A social network has size N if there are N uniquely identifiable agents a_i in A.

This definition of a social network can include any society, so it could be the world wide economy, or Facebook, but from the previous discussions it should be clear that trust and reputation always act in a social network. Without the context of the social network those concepts would be of no use. In most global scale networks we can not assume full observability, therefore Mui defines, with reference to the work of Granovetter [8], the embedded social network.

Definition 2 (Embedded social network)**.** An embedded social network with respect to agent a_i is the unique society of agents A_i that agent a_i is aware of at certain moment in time.

It should be clear from definition 2 that we make no full observability assumption. Each agents acts within the subjective embedded social network.

The social network makes it possible for interactions to happen between agents. We call those interactions "encounters". For the moment we use the simplified definition from Mui and assume that during an encounter both parties chose an action from the set of cooperation and defection: $\alpha \in \{\textit{cooperate}, \textit{defect}\}$. We will later extend this definition to the usecase of Trilber. As explained before (WHERE?) this fits the game of Prisoner's Dilemma which is the theoretical basis for reputation systems in general.

Definition 3 (Encounter)**.** An encounter $e \in E = \alpha^2$ is an interaction between agent a_i and agent a_j such that a_i executed action α_i and a_j executed action α_j.

An agent's encounters are the evidence on which other agents build their opinion of that particular agent and therefore are the building block for trust. An agent's behavior in the past encounters of an define whether other agents will trust that agent or not. Formally Mui defines that history as follows.

Definition 4 (History)**.** $D_{j,i} = \{E^*\}$ is the knowledge that a_j has about previous encounters of agent a_i, which include at least the direct interactions between the two agents but can also include other nteractions of a_i which were "observed" by a_j.

That fact that encounters happen within the embedded social network that connects the two parties in the encounter means that the history does not necessarily include all encounters by a certain agent. With the definition of the history it is possible to define the two concepts of interest, reputaiton and trust. Consider the case that an agent a_i is determining the reputation of another agent a_j which is in the embedded social network A_i. The reputation of a_j in that embedded social network A_i is solely depended on the history $D_{i,j}$, the encounters which a_j took part in and are known to a_i's embedded social network. Mui then defines reputation $\theta_{i,j}$ simply as a value between 0 and 1, where a low value means that a_i thinks a_j has a low intention to reciprocate and a high value means the opposite.

Definition 5 (Reputation)**.** $\theta_{i,j}|D_{i,j} \in [0,1]$ is the reputation of agent a_j as seen by a_i given the history $D_{i,j}$.

Given this definition we are also able to define the "true reputation" θ' which is the reputation as calculated using the complete history of all agents encounters, or $\theta'_j|E \in [0,1]$. Slightly deviating from the model of Mui in order to stay closer to previous work on the specific usecase of the TrustChain architecture we will define reputation as a direct function of this history.

Definition 6 (Reputation function)**.** $R : D \times A \rightarrow \theta^N$ is a function that maps from the known history of encounters D to a reputation value θ for each of the N agents in A.

Finally, the definition of trust as given by Mui is the expectation an agent a_i has that another agent a_j will reciprocate actions in a future encounter.

Given the above definitions a circular relationship between reputation, trust and reciprocity can be induced. Acting reciprocatively in an embedded social network increases an agent's reputation, which in turn increases the trust other agents have in that agent. More trust should then lead to other agent's acting reciprocatively which closes the circle.

However Mui states explicitly that a "decrease in any of the three variables should lead to the reverse effect", thus this circular relationship only holds true if the history of actions is to a large amount transparent to other agents. Also if agents act purposefully wrong and not according to the reputation they calculate the effectivity of the system breaks down. In practice this boils down to problem of a tamper-proof record of encounters and the dissemination of information about those encounters. The next section discusses how this model can be implemented in a system architecture.

3.2. Implementation of the model in TrustChain and Tribler

The definition of the model given in the previous section is from a theoretical point of view for a general reputation system. It is not immediately obvious how this model applies to an Implementation of a distributed reputation system with a specific application. In this section we shall shed light on how the TrustChain architecture and its application context Tribler fit this model. The results are summarized in Table 3.1.

3.2.1. Application context: Tribler

In order to fit the model to the application context of Tribler, which is one context in which TrustChain can be used we have to map the concepts given in the section 3.1 onto the concepts in the torrent client context.

An agent in the model of Mui will generally refer to an instance of the Tribler client running on a machine of a user. A single user can therefore run multiple agents on the same machine. Each instance of the client has a unique identifier. Instances of a client in a distributed system are generally

Table 3.1: Mappings of the theorical model of trust systems to the higher layers of trust systems

Definition	Agent	Encounter	Action	Reputation	Trust
Mui	individuals in a social network	Event between two agents	{ *cooperate*, *defect* }	Value between 0 and 1 showing the perception that suggests an agent's intentions and norms	Expectation that an agent will reciprocate.
Tribler	Instance of the Tribler client	Transaction of data	Two real numbers showing the upload and download during an encounter	Summed upload and download over history	Subjective value calculated by trust function based on reputation
TrustChain					

also called nodes. Encounters between agents, in this case the Tribler clients, are transactions of data on the torrent network or relaying of data for the onion routing. In both cases one agents uploads data and another agent downloads data, so the action space is a real number, where positive values refer to the amount of data uplaoded and negative numbers to the amount of data downloaded.

The mapping of data upload and download is somewhat difficult to map directly to the cooperate and defect actions, and therefore good and bad reputation. In general downloading is seen as consuming value and uploading is seen as contributing value to the network. However Tribler's additional layer of security adds relaying of data as an action to perform and in that case relaying, uploading and downloading the same amount of data, should increase the reputation while downloading more than uploading should decrease the reputation. Qualitatively, agents will have a good reputation if they contribute, that is upload, and relay a lot.

Finally, the difference between trust and reputation is not very straightforward either. However, while the reputation is a well defined number, trust can be seen as a value that is more depended on the network structure. As an example, imagine that we are evaluating our trust in two nodes with a similar reputation, that is a similar amount of uploaded and downloaded data. One node has many interactions with different nodes, most of which are known to us, while the other node has had only a few large interactions with previously unheard of nodes. Taking the definition of Mui as basis for trust, our expectation of reciprocity will be higher for the node that had successful interactions with nodes that we had successful interactions with than for the other node. This fits the definition made in [11], who state that "Trust systems produce a score that reflects the relying party's subjective view of an entity's trustworthiness, whereas reputation systems produce and entity's (public) reputation score as seen by the whole community." Therefore we can define a score, calculated from a certain (subjective) point of view in the network based on the known repuatation of nodes, as trust. Such functions have been defined in previous work, for example NetFlow in [18].

3.2.2. Implementation context: TrustChain

Similar to the application context the model can be mapped to the implementation layer, that is the TrustChain architecture. This way we created a relation between the most basic theoretical layer of reputation systems, the computational definition by Mui, the implementation layer all the way to the application layer of Tribler. This allows for discussions of the problem in the context of each of these layers without loosing the well-definedness property of concepts.

TrustChain is an implementation of a distributed blockchain-based database specifically designed to create trust globally between relative strangers in a digital social network. In TrustChain agents are simply public- and private key pairs. Each agent can be identified by the unique public key which is used in each encounter. Each agent records transactions, in which that agent takes part, as a block on a private chain. The transactions are the equivalent of encounters in Mui's notation.

Definition 7 (Transaction block)**.** A transaction block that describes a transaction between agent a_i and a_j can be defined as a 6-tuple $B^{TX}(t) = \langle\langle tx, pk_i, seq_i, pk_j, seq_j, hash_{B(t-1)} \rangle\rangle$. This is still wrong. We either have to introduce the block proposal/agreement scheme or include two hashes here. where:

- *tx* contains the actions performed during the transaction
- pk_i is the public key of the initiator of the transactions, agent a_i
- seq_i is the sequence number of the block in the history of interactions of agent a_i
- pk_j is the public key of the responder of the transactions, agent a_j
- seq_j is the sequence number of the block in the history of interactions of agent a_j
- $hash_{B(t-1)}$ is the hash of the previous block

As TrustChain is designed to be application agnostic, the possible actions in encounters are not pre-defined but can be anything that can be described by static data. If TrustChain is applied to the Tribler context a transaction block records the amount of data uploaded and downloaded between the two parties of the encounter. Trust and reputation are not directly represented in TrustChain as the system itself is only a way to record encounters, not to interpret them. The interpretation of records of encounters is left to the application context. Still, just like in the trust function defined previously we can assume that in a TrustChain based system the set of encounters, which in TrustChain corresponds to the observed transactions is the single input to the function.

The system does allow to define the embedded social network, the society in which an agent acts as defined by Mui. The embedded social network of an agent a_i in the TrustChain fabric are the agents A_i which agent a_i has directly interacted with, that is the public keys that agent a_i is aware of and knows to exist. This also means that in the case of TrustChain the embedded social network can be described solely by the set of encounters E_i of an agent. However in a global network that embedded social network is usually a very small fraction of the complete social network. This means that chances are low for an agent to be aware of the good behavior of other agents which is one of the fundamental properties that a reputation system needs to fulfill. This brings the discussion to the problem that was described in chapter 2.

3.3. Strategic manipulations

The model of Mui can be mapped onto the TrustChain implementation context in a straight-forward fashion. We showed that in TrustChain, an agent's embedded social network and the agent's true reputation can be inferred simply from the complete set of block that the agent had. Blocks are the receipt of an encounter between two agents.

If the trust system works as expected, a good reputation should have value to agents. All agents on the network attempt to obtain a good reputation and be seen as a trusted partner. As such their behavior should be meticulously agree with the rules. Yet, if the value is large enough and behaving well comes at a large enough cost, agents will aim to bend the rules or even break them in a smart way in order to get the good reputation for free. As a designer of the trust system it is essential to predict the possible ways of manipulation and prevent them. In this section we will define several known types of attacks on trust systems and if applicable define how TrustChain can prevent them.

3.3.1. Forking

Forking is one of the most well-known attacks of blockchain based systems. In the cryptocurrency context it is also known as double spending. As defined in Definition 8 an attacking agent sends two conflicting transactions to two different partners. As long as the two partners are each not aware of the other version of the transaction, both transactions are accepted. This practically allows the attacker to reuse some reputation that normally would already have been lost or overwrite a transaction that would otherwise lower the attacker's reputation.

Definition 8 (Fork)**.** An agent a_i creates two blocks B^{TX} and $B^{TX'}$, with sequence numbers seq_i and seq'_i and partner keys pk_j and pk'_j, respectively. a_i creates a fork if $seq_i = seq'_i$ and $pk_j \neq pk'_j$.

The fact that agents in the TrustChain fabric are solely responsible for their own chain forking is easy and cannot be prevented. However in certain circumstances the attack can be detected, putting a negative mark on the attacker. Namely, if one of the agents who were subject to the attack learn about the other version of the block, they can see that the attacker signed two conflicting blocks. This creates a proof-of-fraud and the agent can ignore the attacker in the future.

3.3.2. Transaction hiding

Once agents have a longer history they will have records of positive and negative encounters. A malicious agent then might try to hide any records of negative encounters, thus boosting the trust other have in the attacker. The architecture of TrustChain is made to make such an attack detectable. The blockchain of the agent creates a tamper-proof, irreversible order for all transactions. Another agent can therefore require the complete sequence of blocks of a partner before interacting. Any missing block will be seen as a malicious request.

The attack can also be mitigated if the partners of the attacker, who also own the blocks that proof the negative encounters of the attacker, disseminate that information to their peers. As such agents may already know about the true reputation of the attacker and decide to ignore that agent in the first place.

3.3.3. Whitewashing

Agents that have lost reputation in a significant amount of transaction may decide to create a new identity for themselves. Thus they can rid themselves from the bad reputation and start fresh. This type of attack is called whitewashing. In the currently deployed version of TrustChain in Tribler this attack cannot be prevented as the software is free and new agents do not need registering with any central institutions.

Still the impact of such attacks can be limited by having some mistrust of new agents joining the network. In that case a new agents need to “pay their dues” before being accepted by the network as equals.

3.3.4. Sybil attack

One of the most serious attacks on trust systems is the Sybil attack. In a Sybil attack the attacker creates a set of new accounts, called the Sybils. Together with the attacking agent the set of agents is called a Sybil region. Those Sybils create transaction records between each other without actually performing the transactions with the goal of boosting the reputation of one of the agents in the Sybil region. The attack is successful as honest agents cannot distinguish between fake and real transaction records. That also makes the attack hard to prevent.

Multiple ways have been explored to prevent this attack. One way is to analyze the network topology and use it to detect Sybil regions. Another way is to increase the cost of registering new identities in the system.

3.3.5. Other attacks

Other obvious attacks are for example a direct tampering with a block. Such attacks can be detected in a straightforward way by simply recalculating the hash and checking with the following block and the signature of the agents.

3.4. Dissemination and verification of records

The possible attacks mentioned in the previous section show that the TrustChain architecture, while allowing honest agents to detect malicious agents, is not secure in a preventive manner. The security relies on honest agents to assemble transaction records of other agents and check them against their current knowledge. We call these two processes dissemination and verification of transaction records. In this section the problems of these process will be discussed that need to be solved in order to fight any malicious behavior in the network.

3.4.1. Complete synchronization and scalability

All of the dissemination mechanisms mentioned in [9] consider the goal of complete information exchange. This would also be a disireable situation in the context of reputation systems. If each agent has knowledge of all encounters, they could calculate the true reputation of all agents and simply agree on whom to trust. The system would also be secure against malicious behavior.

However, a global reputation system that tracks all interactions of all agents creates a huge amount of data which would need to be transmitted and stored on all agent's devices. This seems like an unfeasible target. Instead a “high level” of information dissemination should be strived for.

It should be clear the whether the state of complete synchronization can be achieved depends on the rate at which new interactions are recorded and the rate at which information dissemination happens. In fact, Bitcoin achieves full synchronization (except for the newest blocks which are seen as “not yet confirmed”) through restricting the block time and size. With a block size of 4MB and a 10 minute block time, there is enough time between new blocks such that nodes can synchronize with the updated state of the system.

3.4.2. Second-order manipulative behavior

Another intricacy of distributed system is that the designer of the system has no control over the actual behavior of agents. That is why incentives need to be put in place in order to make it disadvantageous to deviate from the designed behavior. Given the right incentives and rational agents that try to maximize their value function the designer can be sure that no misbehavior will spread as it would decrease the value function of those agents.

The problem with the TrustChain system is that while it's security and validity as a trust system rely on the dissemination and verification of data, there is no incentive in place to guarantee that agents engage in any such activity. Actually the opposite is true when assuming that data storage, computation power and bandwidth are costly resources in the eyes of agents. It is actually disadvantageous for agents to observe interactions of other agents, store them and verify them against their previous knowledge in order to detect any malicious behavior. It is therefore possible to free-ride, not in the context of an application like Tribler, but in terms of information dissemination: agents can decide to not share and verify information and still take part in the network as valid agents.

We can thus define another class of malicious behavior which we will call second-order manipulations which is distinct from those mentioned in Section 3.3, which we will refer to as first-order manipulation. Agent behavior that does not immediately violate any rules for data creation but rather does not help the network defend itself against attackers falls into this new class. Specifically we define two such manipulations: dissemination free-riders and verification free-riders.

Definition 9 (Dissemination free-rider)**.** A dissemination free-riders behave honestly except that they do not actively distribute any transaction records to peers.

Definition 10 (Verification free-rider)**.** A verification free-riders behave honestly except that they do not verify any data that they receive from the peers.

Both types of second-order manipulators attack the security of the network through passively not helping to defend it. The immediate effects of single agents behaving in this manner are small to non-existent; the network security is held up by the honest majority and attackers will still be identified. The second-order manipulators ride free on the security provided by their peers. However, should a large fraction of the network realize that free-riding is possible the security could be seriously be harmed.

3.4.3. Creating incentives

In order to solve the problem we need to realize how incentives can be created. Bitcoin solves the incentive for sharing transaction blocks by given nodes a reward for mining a block. Only if the node broadcasts the block, can the award be claimed. Also other nodes want to stay updated on the state of the chain in order to mine a block on the most recent chain as new blocks for shorter chains will not be accepted.

Instead of giving awards in order to encourage behavior, we can also punish nodes in order to discourage bad behavior. For example, double spending is discouraged because the records of transactions make the attack detectable such that other agent can punish the attacker.

Another way would be to combine the sharing of information with the reputation system built on top of TrustChain. In that way good reputation can not only be built through behaving well in the application context but also by being a good agent in the TrustChain network. Helping the network to defend against malicious nodes by obtaining and spreading information should then be rewarded.

All ideas require a feature that TrustChain does not offer at the moment: the recording of exchanges of information. All we have presented before about TrustChain is concerned with transaction in the application layer but not the record layer. However, in order to reward the exchange of information or punish free-riding on this layer, this information needs to be stored in a tamper-proof manner just like the transactions. That is the state of an agent, which we argued can be described by the encounters that the agent had in the past, should include also the information exchange behavior.

4

Internal agent state transparency

In the previous chapters we analyzed the problem of information dissemination of data for a distributed repuation system built on the architecture of TrustChain. We have argued that in order to make the dissemination strategy proof, which is neccessary to guarantee that no agent deviates from the desired behavior without being detected and properly dealt with, we need to publicly record the action of sharing knowledge. In this chapter we propose an extension of the TrustChain architecture that enables strategy proof information dissemination and validation in a distributed TrustChain based network without deminishing the scalability properties.

4.1. Concept proposal

In the previous chapter we formally defined the internal state of an agent as not only the encoutners of agent's own encounters but the complete knowledge of the network the agent has. That knowledge can be represented by the set of encounters the agent is aware of. Based on this we can define a desired property that a fabric like TrustChain needs to fulfill in order to provide strategy-proof sharing of information.

Definition 11 (Interal agent state transparency)**.** The internal agent state is transparent, and therefore this property is fulfilled, iff:

- an agent a_i can require an agent a_j's internal state from any point in time
- an agent a_i can determine whether an agent a_j is lying about her internal state

When the property is fulfilled there should be an exchange protocol after which an agent is aware of all information the other agent has as well as a verfication protocol which validates that the claimed internal state is indeed valid and complete.

In order to achieve internal agent state transparency for TrustChain, any exchanged information needs to be public. We will later describe the incentive for agents to actually perform this exchange. So, conceptually a record exchange can be defined as follows.

Definition 12 (Record exchange)**.** A record exchange X_k of size m contains a list of transaction blocks $X_k = \{B^{TX}_{k,0}, B^{TX}_{k,1}, \dots B^{TX}_{k,m}\}$.

Finally, the complete state can be described by the union of performed transactions and transactions obtained through exchagnes.

Definition 13 (Internal state composition)**.** The internal state S_i of agent a_i with true transaction history D'_i and k exchanges can be inferred as follows.

$S_i = D'_i \cup \{X_{i,0} \cup X_{i,1} \cup \dots X_{i,k}\}$

4.2. Implementation of internal agent state transparency in TrustChain

In the previous section we conceptually defined the internal agent state transparency for TrustChain. This section will describe how those concepts can be implemented in a working distributed software system.

First and foremost, a new type of record needs to be created in which no application specific transactions, but system transactions of other records will be documented. We therefore extend the TrustChain architecture (described in detail in WHERE?) with exchange blocks

Definition 14 (Exchange blocks)**.** An exchange block $B^{EX} = \langle\langle ex_{\text{up}}, ex_{\text{down}}, pk_i, seq_i, pk_j, seq_j, sig_i, sig_j, hash_{B(t-1)}\rangle\rangle$ is defined as a tuple, where:

- ex_{up} is the top hash of the list of hashes of blocks the initiator a_i shared with a_j
- ex_{down} is the top hash of the list of hashes of blocks the responder a_j shared with a_i
- pk_i is the public key of the initiator of the exchange, agent a_i
- seq_i is the sequence number of the block in the history of interactions of agent a_i
- pk_j is the public key of the responder of the transactions, agent a_j
- seq_j is the sequence number of the block in the history of interactions of agent a_j
- sig_i is the signature by the initiating agent a_i
- sig_j is the cryptographic signature by the responding agent a_j
- $hash_{B(t-1)}$ is the hash of the previous block

Each exchange block describes a pairwise exchange of blocks. The exchange is deliberately made bidirectional in order to provide an incentive for both agents to sign the exchange and keep to the promise of publishing the exchange on their chain. Instead of publishing the exact blocks exchanged, a list of block hashes is created and the hash of that list is published in the exchange block. One hash for the block transferred from agents a_i to a_j and another hash for the blocks transferred in the other direction. This reduces the amount of data put directly on the chain, however it makes it impossible to infer the internal state of the agent from the chain only. That is why each agent internally needs to keep track of the actual content of the exchanges.

Definition 15 (Exchange storage map)**.** Each agent keeps track of the content of all exchanges with an exchange storage map $F : hash_{B^{EX}(t)} -> X(t)$ which maps the hashes of exchange blocks to the respective record exchange, so the list of blocks the agent received in the exchange described by $B^{EX}(t)$.

The combination of exchange root hashes and the exchange storage kept by each agent enables tamper-proof exchange of information. This enables agents to determine the state of another agent. Consider an agent a_i that tries to determine the state of agent a_j. Agent a_i requests both the chain and the exchange map from agent a_j. If a_j does not respond, agent a_i is not able to determine the state and has to assume that a_j does not respond in order to hide malicious behavior. In that case a_i will add a_j to the list of blocked agents.

If both the chain and exchanges are available, the agent a_i loops through the chain. Each block of the chain is added to the state and if the block is an exchange block, also all blocks that were transmitted during the exchange documented by that block, which can be looked up in the exchange map are added to the state. The result is the complete internal state of agent a_j. The algorithm is described formally in algorithm 1.

Algorithm 1 Determining the internal state of another agent

procedure determineState
  a_i : Initiator
  a_j : Responder
  chain ← *request_chain*(a_j)
  F ← *request_exchanges*(a_j)
  state ← []
  if not *chain* or not F **then return** false
  for all *B* in *chain* **do**
    if *B* is an exchange **then**
      state = *state* $\cup F(B)$
    state = *state* $\cup\{B\}$
  return *state*

4.3. Making sharing strategy proof

How is strategy proof even defined? We have described how the internal agent state transparency property can be added to the TrustChain architecture, however this in in itself does not make dissemination of information strategy proof. It only creates a tool to document information exchanges in a tamper-proof manner. The next step is to use the knowledge about information exchanges conducted by the agent to make decisions about future interactions.

One solution is to build a system internal reputation which is based on the amount of data shared with other agents. Agents that exchange more data with other agents and initiate exchanges to obtain more data should have increased reputation. Helping the system defend itself against attackers and making information widely available would then be advantageous as other agents see the good behavior, have a better opinion of that agent and share data with them.

Another solution is to define policies that require a certain amount of data exchange for each transactions. Let's say that the system designer introduces a policy which requires honest agents to only interact with other agents that have at least one data exchange for each transaction they conduct. This is simple to verify: the agent scans through the chain and counts the transaction and exchange blocks and verifies that there are at least as many exchange as transaction blocks. Otherwise the agent will simply not interact with that agent.

4.4. Possible ways to work around the security

In this section a few examples of maliciously acting agents will be analyzed theoretically. The same will be done in experimental analysis with a proof-of-concept. We assume that all agents that interact with each other also verify each other's chain and internal state.

First of all, what the original TrustChain implementation allowed was to free-ride on information sharing and verification. That is, even though honest nodes were polling other agents for information on transactions and verified that no double-spends happend, other nodes were able to not do that and still interact with honest agents.

Definition 16 (Lazy free-rider)**.** We define a lazy free-rider as an agent that performs normal in transactions but never performs an exchange of information on transactions.

In a network of honest agents, the lazy free-rider will quickly be found and blacklisted as that agent will simply not have any exchange blocks in the chain.

A more elaborate way of free-riding is to exchange blocks and create exchanges but delete the data as soon as it arrives.

Definition 17 (No storage free-rider)**.** A no storage free-rider is an agent that does exchange data but does not store the received data.

A no storage free-rider can only commit a few interactions before being detected as a fraud. Let's consider the first verification. The no storage free-rider agent only has the genesis block which is cheap to share. Therefore the first verification will go through and an exchange block will be created. However, the second verification is more difficult to pass as some information will have been received as recorded in the exchange block but cannot be conjured on request. Any further verifier should therefore find that the agent is not able to provide the blocks of the first exchange recorded on the agents chain.

4.5. Making verification strategy proof - Verification of verifiers

Another way of free-riding on the original TrustChain architecture is to not actually validate any data. Even if we solve sharing of data, agents can still save computing power by not scanning their peers for malicious behavior in the hope that other agents on the network do perform the validations and will inform them once they find something. The new extension makes it possible to detect this behavior in case they fail to report a malicious transaction.

Definition 18 (Validation free-rider)**.** An agent that does perform honestly in terms of transactions and exchanges but does not validate the behavior peers is called a validation free-rider.

The proposed extension of TrustChain enables an agent a_i to determine the internal state of agent a_j, that is obtain the list of all blocks of a_j. Consider that a_j is a validation free-rider and a_i has obtained a_j's internal state. If a_j has obtained the knowledge of a malicious transactions, for example two conflicting blocks of a double-spend, a_j will not have realized it as the validation has never been performed. However a_i is aware of all blocks that a_j has and therefore also aware that a_j should be aware of the conflicting blocks if a_j was honest. Now, a_j's free-riding will be detected if a_j has performed an interaction with the agent resopnsible for the conflicting transactions. That is because any honest agent would have detected the conflict and ignored that agent except for colluters or validation free-riders.

4.6. Considerations of asynchronous transactions

In the previous discussion we have actually simplified the system to one in which each agent only performs one transaction at a time. This makes it more straight-forward to understand the properties and see that the extension does provide a strategy-proof dissemination. In a real-world system, depending on the use case, a user might need to quickly engage in multiple transactions and communication delays do not allow to finish transactions first and then continue with the next. In previous work we have shown how TrustChain can work in an asynchronous way, by splitting the transaction blocks into a block proposal and a block agreement. This also creates additional complexity for the exchange blocks as the asynchronous reception of blocks needs to be recorded in order to keep the property of internal agent state transparency.

Definition 19 (Async exchange block)**.**

5
State consensus through anti-entropy

In the first chapters of this thesis we introduced the incentive problem of information dissemination in distributed reputation systems. In the previous chapter we introduced an extension of the TrustChain architecture that allows for agents in the network to obtain and verify each other's internal information state. But in order to achieve the goal of strategy-proof dissemination of information a tangible mechanism is required that applies the advanced tools that the new architecture provides. In this chapter we analyze one such mechanism which is based on the anti-entropy concept. We first describe the mechanism conceptually, then discuss the implementation details and finally elaborate on some intrinsic properties the mechanism could introduce in actual applications. In the next chapter an implementation will be analyzed experimentally with small agent sets in order to prove that properties from the theoretical analysis can be observed in practice.

5.1. Conceptual description

Section 4.3 explained that the architecture itself does not solve the incentive problem. It rather provides the evidence on which an incentive mechanism can be built. Many different mechanism are possible, depending on the specific needs of the application context. This offers a lot of flexibility for system designers which is different from existing architectures which a very static in their protocol and have pre-defined security and scalability properties.

5.1.1. Design choice: security vs scalability

Security, decentralization and scalability are three properties that are traded against each other in the design of a decentralized system. It can be argued that TrustChain was designed with scalability as the highest priority while Bitcoin was designed with security as highest priority property. We argue that the extension that allows for internal agent state transparency allows for flexibility in the design choice of security and scalability. In this section we propose a mechanism that trades some of the scalaiblity of TrustChain for stronger security in order to show that a secure mechanism is possible on top of the TrustChain architecture.

5.1.2. Concept: anti-entropy

The mechanism is based on the concept of anti-entropy, which was described in [7] for the purpose of maintaining mutual consistency between multiple replicas of a database. Updates to the database can arrive at any single site and need to be forwarded to all other replicas. Demers et. al. study three different mechanisms to disseminate the updates to other sites: direct mail, anti-entropy and rumor mongering.

In the direct mail mechanism, a database forwards the update to all other database immediately, which seems like the most straight forward approach but is restricted by the fact that each database does not know about all other databases.

Anti-entropy is a process in which each database periodically chooses a partner database and both exchange all database contents in order to resolve any differences between the two. The process was found to be reliable but slower than direct mail.

The final mechanism is called Rumor Mongering. Sites consider updates “hot rumors” after receiving them for the first time. While the site considers the rumor “hot”, it choses periodically another site and informs it about the rumor. When the site has encountered a certain amount of sites which were already aware of the “hot rumor”, that update is retained but site stops with actively propagating the update.

For the purpose of this work, we will focus on the concept of anti-entropy. Direct mail is not a viable option because for large social networks the embedded social networks are a very small subset and the rest of the agents in the network would not be informed of updates. Rumor mongering effects are best observed in larger networks however this work is concerned with conceptual analysis and the experimental analysis in the next section concerns small networks. Also there are more algorithms than these three but anti-entropy fits the architecture of TrustChain very well and is a good starting point for analaysis of dissemination mechanisms. The analysis of other mechanisms will be subject of future work.

5.1.3. Replicated databases vs TrustChain

The context of the work of Demers et. al. is similar to the context that this work is concerned with in many regards. Replicated databases are a distributed system as all instances of the database are independent, equal entities, just like the agents in the TrustChain network. Each agent has an internal state which is equal to the set of transactions that agent is aware of which is equal to the state of the database which is equal to the entries that database is aware of. Our goal is to propagate information on new transactions just like the goal of Demers et. al. was to propagate updates to the databases.

In the context of reputation systems anti-entropy allows for two agents to align on their knowledge of the social network, that is to obtain the same embedded social network and agree on the reputation of all agents in that network. Two agents, a_i and a_j have two different internal states, represented by the sets of encounters E_i and E_j. There can be some overlap between the two sets, but that is not guaranteed. Agent a_i chooses to synchronize states with agent a_j. Both agents send their own set of known encounters and merge them. This results in a new set $E_{i,j} = E_i \cup E_j$. In the context of TrustChain this translates to the exchange of transaction blocks, such that after the exchange both agents have the exact same set of transaction blocks. As the reputation of peers is calculated from the set of transactions both agents agree on a single reputation vector. If the two agents also use the same function for trust calculation both can even agree on a single trust vecotr. The two agents have reached consensus on trust and reputation.

The exchange of information will be recorded in the form of exchange blocks on the chain of both participating agents as explained in the previous chapter, section 4.2.

The consensus is only reached at one point in time and is not maintained. Once any of the two agents conducts another anti-entropy exchange or a transaction, the other agent is not required to be informed or to observe the interaction. That is after the exchange both internal states can diverge until the same two agents happen to perform another state synchronization.

In the work of Demers et. al. database instances chose partners for anti-entropy exchanges at random which is a valid strategy as each peers updates seem equally important. In contrast, reputation systems should value the information about possible future interaction partners as more important. Therefore, our proposed mechanism requires agents to at least perform an anti-entropy exchange with those agents that they will have an interaction next. That way, both parties of a transaction are required to obtain and verify each others information in order to make sure that the transaction will be done on top of a valid state. If any party does not agree with the state of the opposite party, the transaction will not take place. If any party performs a transaction eventhough the information clearly shows misconduct on the part of the partner, they will also be held responsible for not performing their validation responsibility. Without the requirement of validating transaction partners, agents can purposefully exchange data with honest agents but perform interactions with dishonest agents and later claim to have had no knowledge of the dishonesty of the partners.

5.2. Implementation of anti-entropy exchanges

In the previous section we described the anti-entropy method for information exchanges between agents. This section elaborates on the implementation details of the mechanism in the TrustChain architecture. We start with an application agnostic example of the exchange of information and the transaction process.. In the next section we expand on the considerations for application specific implementations.

We consider an agent a_i, who is about to conduct another interaction. What the interaction is about and how the partner is chosen are specific to the application context. For this example we assume that a_i can randomly choose an agent from the network. Once a_i has chosen a partner a_j, a_i starts the communication and is therefore the *initiator* while a_j is the *responder*.

The initiator starts the interaction by sending the chain and exchange history to the responder. The chainIs this explained somewhere includes the blocks that describe the transaction and exchange history Is this explained somewhere of the initiator while the exchange history includes the index of blocks exchanged for each exchange block. On reception, the responder can verify the chain using the algorithm 2. The algorithm first checks, wether the chain is a complete sequence without missing blocks in between. If the check is positive, the number of exchange and transaction blocks is compared, as well as the public keys of partners such that each transaction can be paired with a succesful exchange previous to the transaction.

Algorithm 2 Chain

1: **procedure** verifyChain

Once that check also succeeds, the responder is able to build a block index that indexes the internal state of an agent. The block index is a summary of the contents of the internal state and shows which transactions of which agents are knwon to the agent. This is an optimization that allows agent to request only specific blocks instead of the complete database of another agent. This is a lot faster if agents that already share a lot of data. Agent a_j compares the calculated block index with his own index and request the difference in blocks, so those that a_i has but a_j does not have from a_i.

The initiator receives the request and replies with the blocks that a_j requires to perform the complete internal state validation. Should the responder, for whatever reason, wrongly require more blocks, so also blocks that are not in a_i's posession, the interaction will be canceled.

The responder receives the missing blocks. At this point a_j should be in the posession of all of a_i's blocks plus some blocks that a_j has over a_i. Agent a_j is then able to perform the complete internal state verification according to algorithm 3. A state is valid if:

Algorithm 3 Chain

1: **procedure** verifyChain

- the chain is valid as per algorithm 2
- the hashes of the blocks according the the exchange indexes match the hashes recorded on the chain's exchange blocks
- Any recorded misbehavior of other agents is reflected by those agents' public keys in the ignore and block list

If the internal state of a_i is determined to be valid by a_j, the responder shows approval by sending the own chain, exchange history and blocks (which can be calculted by taking the opposite difference from before) to agent a_i.

Once the initiator receives that data from the responder, the second validtion of chain and state, this time agent a_j as subject, can be conducted. If also this checks out, the valdiation phase is completed and the initiator can publish an exchange block proposal, which includes the hashes of the exchanged blocks. The responder receives that proposal and the hashes contained in the proposal block match the hashes that a_j calculates for the excahnged data, the block is signed and returned.

This concludes the anti-entropy exchange, after this the two agents perform a normal interaction.

5.3. Example

In order to make the mechanism clearer, an example is provided in this section. We look at two agents, Alice and Bob. Alice wants to interact with Bob and starts the interaction. We assume that both agents are new to the network and only have their genesis blocks on the chain. In order to start the interaction, Alice sends her chain (only the genesis block) and an empty set of exchanges to Bob. Obviously the genesis block is accepted and Bob shows his approval by sending his own genesis block and an empty set of exchanges. Also Alice accepts the data and creates an exchange block proposal. Figure 5.1 shows the chains of Alice and Bob after the complete interactions. The block proposal is Alice's block 2. When Bob receives the block, he checks that the exchange hashes in the proposed exchange block match the hashes of the two genesis blocks from Alice and himself. If he agrees, he documents the acquisition of the exchange block in a single-signed exchange block (Bob's block 2) and creates the agreement block (Bob's block 3) to the exchange block proposal. He sends back the exchange block agreement to Alice, who herself creates a single-signed exchange block (Alice's block 3) for the acquisition of the exchange agreement block. At this point Alice is ready for the actual transaction. How the transaction happens exactly is not of importance for this example. After the transaction was successfully conducted it also needs to be documented. Again, the same process as with the exchange is followed. Alice creates a transaction block proposal (Alice's block 4), Bob creates the acquisition block and the agreement block (Bob's block 4 and 5). Finally, Alice creates the acquisition block of the agreement block of the transaction.

This example sheds light on the block creation process but is too simple to properly explain the exchange and verification process. We extend the example with another agent, Charles, whom Bob would like to interact with after the previous interactions. Again, Charles is assumed to be new to the network so the genesis block is the only block on his chain. Bob starts the interactions by sending his chain and exchanges. Now Charlie performs some checks:

- *Is the chain complete?* - the chain is a full sequence (1 through 5) and Charlie is not aware of any later blocks than 5, so the chain seems valid
- *Are the exchanges correct?* - there are three exchange blocks on the chain, one receiving the exchange block proposal, one for the actual exchange of the genesis block of A and one for the transaction block proposal. Charlie recalculates the hashes stored in the exchange blocks from the blocks sent by Bob. If all the hashes are equal, the exchanges are accepted.

Once Charlie accepts all the checks, he sends his own chain and exchanges which are checked by Bob and the interaction continues as previously described. Table 5.1 shows the blocks that each agent has after the first and second round. After the second round, Charles has all the blocks from Bob that he had after the first round.

Table 5.1: The block databases of each agent for the example

Database	Alice	Bob	Charles
Before first round	A: $[1]$	B: $[1]$	C: $[1]$
After first round	A: $[1, 2, 3, 4, 5]$ B: $[1, 3, 5]$	A: $[1, 2, 4]$ B: $[1, 2, 3, 4, 5]$	C: $[1]$
After second round	A: $[1, 2, 3, 4, 5]$ B: $[1, 3, 5]$	A: $[1, 2, 4]$ B: $[1, 2, 3, 4, 5, 6, 7, 8, 9]$ C: $[1, 3, 5]$	A: $[1, 2, 4]$ B: $[1, 2, 3, 4, 5, 6, 8]$ C: [1, 2, 3, 4, 5]

In order to make even clearer how the implementation works internally, Table 5.2 shows which mappings of exchange blocks to exchanged blocks exist.

5.4. Consideration of application specific features

choosing partners – we now chose partners randomly but probably for many applications there are better strategies application specific rules – we could perform additional verification for application context, for example no downloading after a certain negative balance boundary

5.5. Advanced implications of anti-entropy

locality – when you need to exchange all information and storage has value, it is cheaper to interact with agents that have a largely similar information set sybil attack – when we can apply application specific

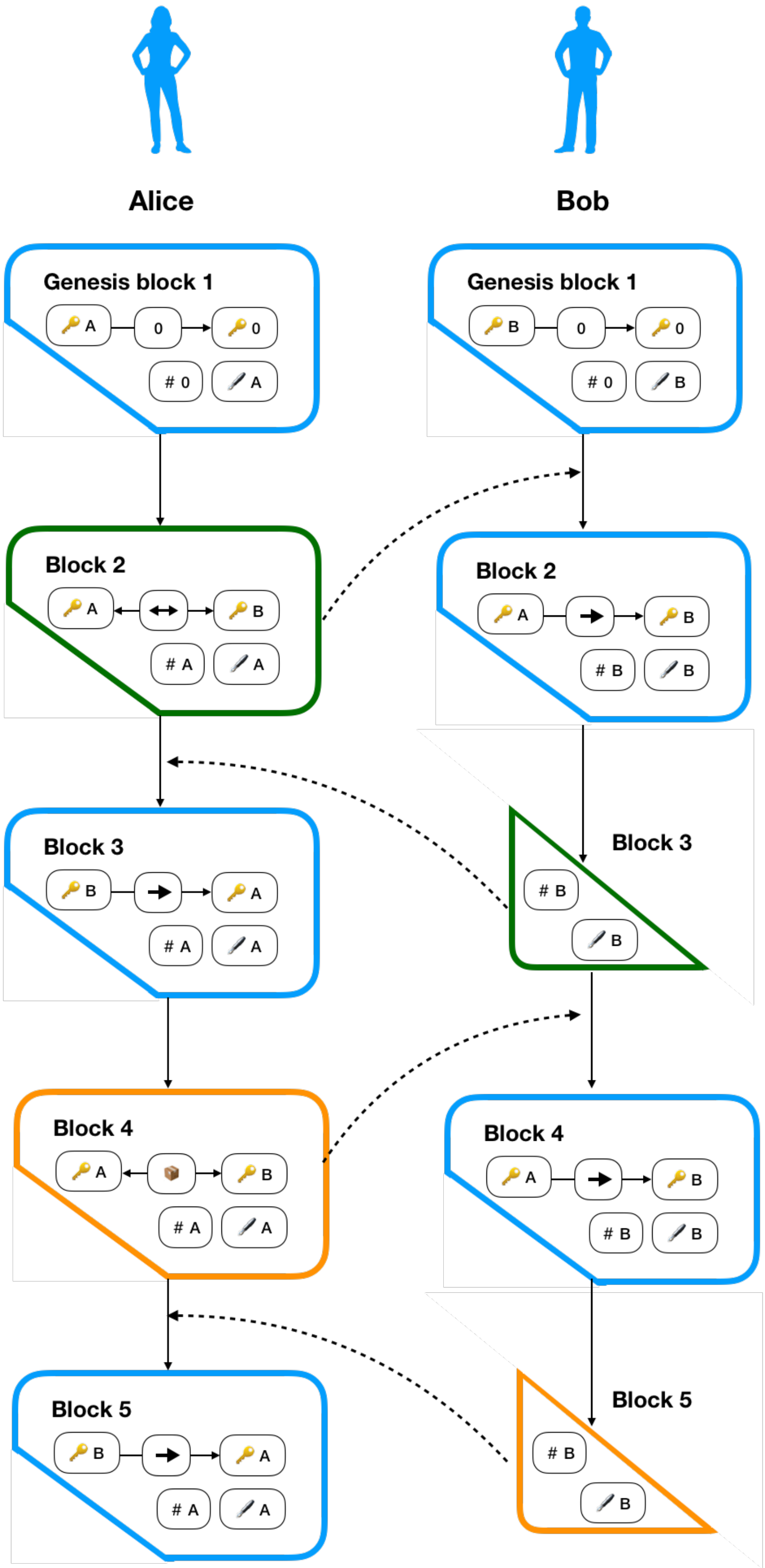


Figure 5.1: Example of one interaction between two agents

Table 5.2: All exchange blocks with their corresponding exchanged blocks

Block	Exchanged blocks
Alice Block 2	B: [1]
Alice Block 3	B: [3]
Alice Block 5	B: [5]
Bob Block 2	A: [2]
Bob Block 3	A: [1]
Bob Block 4	A: [4]
Bob Block 6	C: [1]
Bob Block 7	C: [3]
Bob Block 9	C: [5]
Charles Block 2	B: [6]
Charles Block 3	A: [1, 2, 4], B: [1, 2, 3, 4, 5]
Charles Block 4	B: [8]

rules and we require agents to perform checks of all agents they interact with, we can introduce a rule that says agents cannot upload to agents that are completely new to the network (they first have to prove themselves). This way an agent cannot create fake agents that all download from one agent and thus increase that agents balance without actual downloading happening

6
Experiments and results

Theory shows that internal agent state transparency and the anti-entropy mechanism reward honesty and punish any strategic manipulation. We implemented this mechanism and establish in this chapter how it handles strategic manipulation. More specifically we emulate honest agents and multiple types of strategic manipulators in small scale experiments. We can observe the behavior of the honest agents and find that strategic manipulators are detected and isolated. We show that honest agents who execute our mechanism are able to effectively detect malicious agents that do not share or do not verify their partners and ignore them for future interactions.

The rest of the chapter is structured as follows: we first give an overview of the software architecture. Then we explain the setup of the experiments and the types of strategic manipulation that will be emulated. Finally we present the results of the experiments.

6.1. Experiment design

The goal of the experiments is to prove the true effectiveness of the mechanism at detecting manipulation attempts and isolating malicious agents. Three different types of dishonest behavior will be analyzed in the experiments.

- **Dissemination free-rider**: An agent that gains an advantage by not expending resources on disseminating transaction records.
- **Verification free-rider**: An agent that gains an advantage by not expending resources on verifying the behavior of their peers
- **Malicious**: An agent that manipulates or withholds information in order to gain an advantage.

In the experiments we emulate small agent networks up to 6 agents. We assume that agents are trying to perform interactions with each other. The experiments are not concerned with the actual trust agents have in each other so we keep transaction blocks empty. Agents are acting completely autonomously but knows about all the other agents in the network. At a frequency of 20 per second agents go through rounds, in each round an agent has a 1% chance of starting an interaction. This adds up to approximately 1 transaction every 5 seconds. In addition agents respond to interaction requests from their peers asynchronously. At the start of the interaction a peer is selected with uniform probability. If an interaction with the selected partner is already ongoing, the new interaction request is cancelled without selecting a new partner. The same happens when the selected partner is a known malicious agent. Once the interaction is started honest agents perform according the mechanism as described in chapter 5. The other types of agents each have some deviation from the expected behavior in order to obtain an unfair advantage. All types of agents which were used in the experiments are listed in Table 6.1.

With these types of agent we run experiments with different sets of agents. In experiments with only a single dishonest agent, the experiment is successful if the honest agents stay among themselves and ignore the dishonest agent. That is, the dishonest agent should have 0 transactions at the end of the

Table 6.1: Agent types used in the experiments

Type	Sub-type	Behavior
Dissemination free-rider (DFR)	No exchanges	Does not create any exchanges
	Empty exchanges	Creates exchanges blocks with empty exchanges
	Self-requests	Exchanges only own chain
Verification free-rider	-	Acts honestly but blindly trusts all partners without verifying their data or behavior
Manipulator	Hiding transaction	Creates a normal transaction and tries to hide it afterwards
	Forking	Creates two conflicting transactions and shares them with two different peers

experiment. In the case with multiple dishonest agents we have to make a distinction between multiple single acting dishonest agents and collaborating groups of dishonest agents.

6.2. Implementation details

The experiments are run on a standalone implementation of TrustChain with the mentioned extension and mechanism. The code is available through GitHub[1]. The code is based on the TrustChain implementation of py-ipv8[2] another project that is developed in the context of BlockchainLab.

The implementation is done in Python. The programming language was chosen as it allows for fast development, offers many useful extensions and the py-ipv8 dependency is also written in Python. At the core of the software is the `agent` module. It implements the agent base class which contains all functionality for communicating with other agents. They communicate with each other using tcp sockets, implemented using the `zeromq` library. Messages for the communication are defined in the Google `protobuf` format. The basic TrustChain data structures and block database implementation were taken from the `py-ipv8` project. Next to the agents there is the discovery server which handles the peer discovery process.

The experiment is started by spawning all agent processes and the discovery server. Once the agent process is started each agent sends a registering message to the discovery server to register the public key with the address of the tcp endpoint. After a 5 second initialization period it is assumed that all processes have started and registered. The discovery server then sends a message to all registered agents containing their peers. Once that messages is received agents start running the experiment.

6.3. Experiment results

In the following we will present the results of our experimental analysis.

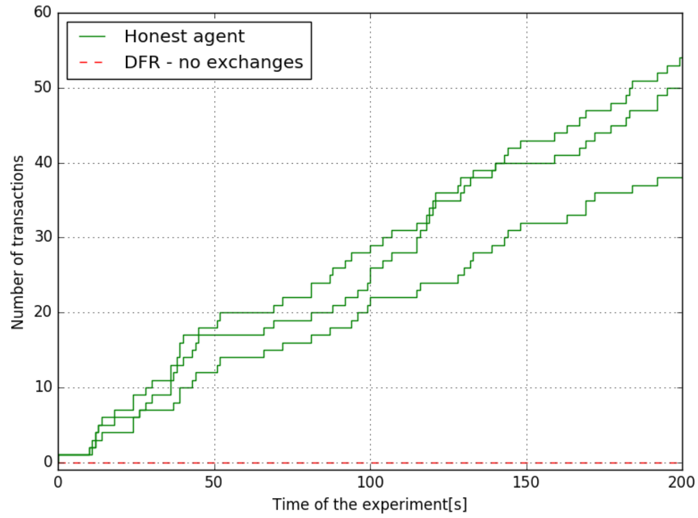
6.3.1. Dissemination free-riders

The first experiment to run is concerning the dissemination free-riders. One desired property of the internal agent state transparency is that disseminating information is strategy-proof. That means, it should not be advantageous to not share information. In this first set of experiments we observe the behavior of three honest agents in the presence of three different types of dissemination free-riders. Figure 6.1 shows the number of transactions of each agent against the time of the experiment.

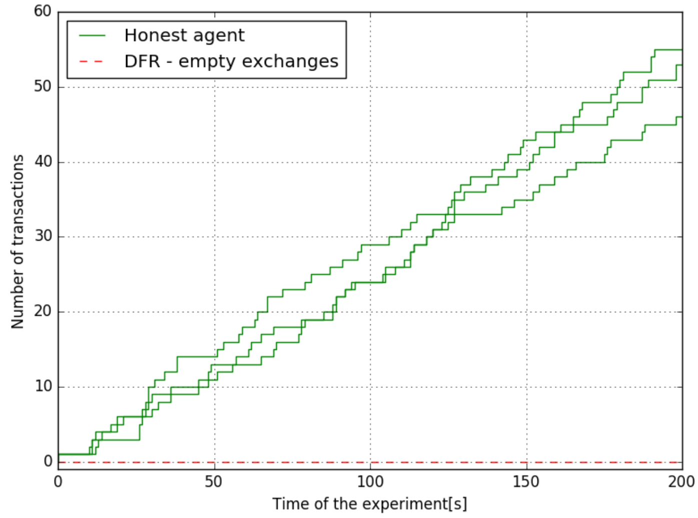
All experiments show the same general picture: all honest agents steadily increase their succesful interactions throughout the experiment and end up between 40 to 60 transactions. The dishonest agents are not able to perform a single interaction. As expected the dissemination free-riders have no chance to interact with the honest agents in any of the three cases. That means the honest agents detect the wrong behavior of their malicious peers and ignore them for future interactions.

In the first experiment 6.1a the dishonest agent does not exchange any data and therefore does not create any proof of exchanges. Yet in order to interact with honest agents the dissemination free-rider

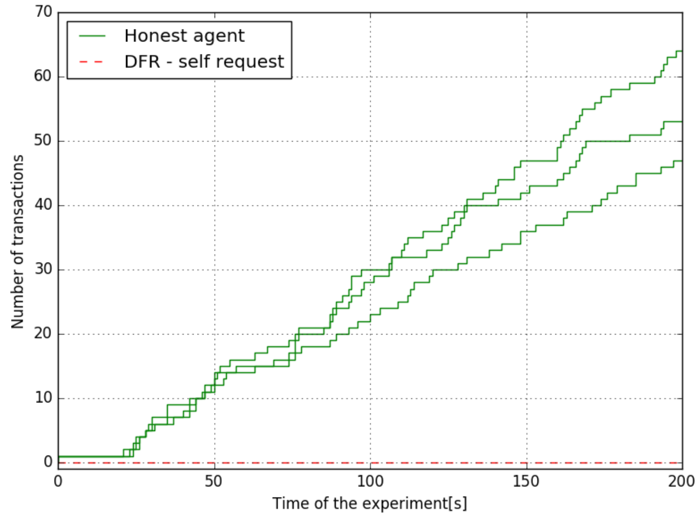
[1] https://github.com/jangerritharms/aupair
[2] https://github.com/tribler/py-ipv8



(a) Transactions over time of honest agents with dissemination free-rider that does not perform any exchanges



(b) Transactions over time of honest agents with dissemination free-rider that creates empty exchanges



(c) Transactions over time of honest agents with dissemination free-rider that only exchanges own data

Figure 6.1: Transactions over time of three experiments with different types of dissemination free-riders

needs to publish a complete chain. Honest nodes detect the lack of exchange blocks upon inspection and subsequently distrust that agent.

The second experiment 6.4b the dishonest agent aims to create empty exchanges. In the role of the responder, the agent requests no blocks from the honest agents and as initiator the agent tries to claim that no blocks were received from the honest agent. However, as explained in the theory section, empty exchanges are not possible because agents always have at least one new block to talk about. Also agents only sign an exchange block if they agree with the hash of transferred blocks. That is, if the dishonest agent claims no blocks were sent but actually the honest agent did send data, the honest agent will not sign the block. That way, also in this case the honest agents are able to detect the wrong behavior.

Finally, the third type of dissemination free-riders, results shown in Figure6.1c , requests and sends data about itself but not any other knowledge. When in the responder position, the agent can request only its own blocks. As initiator the agent tries to act again as if no data was received other than its own blocks. However the partner can see that the exchange block proposal does not properly represent the data exchanged and will therefore stop the transaction. Also this way the agent is not able to interact with honest partners.

We find that dissemination free-riding leads to isolation and no transaction with honest partners. That means no reputation or trust will be build with this type of misbehaving agents and any hope for future rewards is voided. Agents have to disseminate their data in order to be accepted and to prosper.

6.3.2. Collaborating dissemination free-riders

The situation changes when multiple dissemination free-riders join forces and collaborate in creating a subnetwork. This situation can arise if a software is forked and an alternative release is published. In order to analyze this situation we ran an experiment with three honest agents and three dissemination free-riders. The transactions are plotted against time in Figure 6.3.

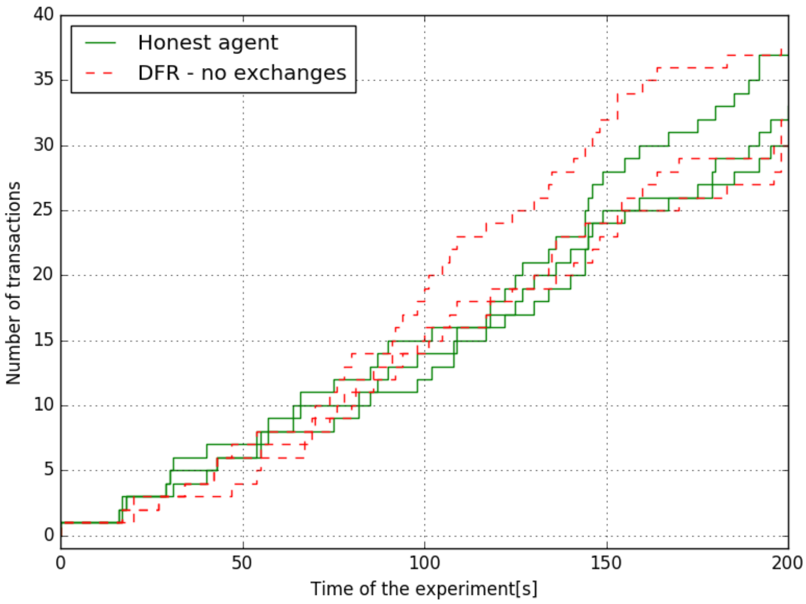


Figure 6.2: Transaction history of three honest agents and three dissemination free-riders that are cooperating

Again each line in the plot refers to the successful transactions of one agent. The plot shows six lines going almost in parallel which means that all agents fare equally well. From the graph it is not immediately obvious how this is possible as in the previous experiment, dissemination free-riders were not able to get any successful transaction. Therefore an interaction matrix is shown in Figure 6.3 which shows for each agent, how many interactions they had with each of their peers. The plot shows fields with many interactions between the honest agents as well as for the dishonest agents among each other. However the intersection of the two groups only shows blue fields and zero interactions.

The two plots together shows that dishonest agents can collaborate and prosper next to the honest

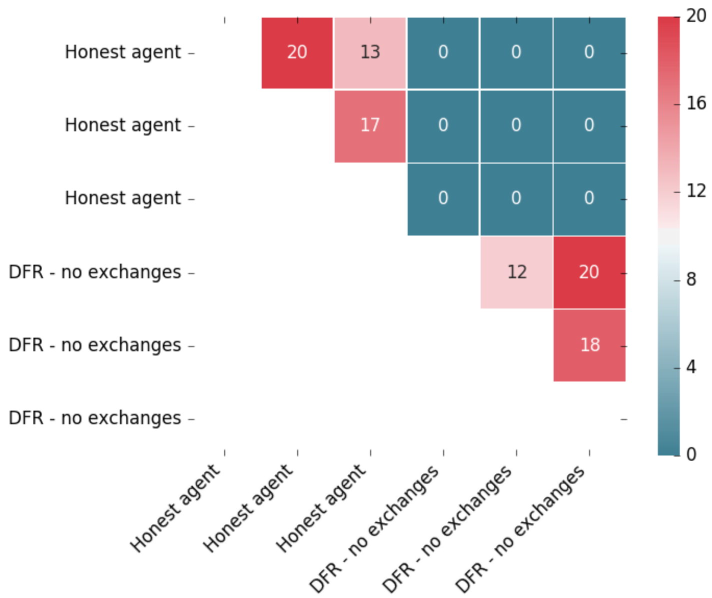


Figure 6.3: Interaction matrix of three honest agents and three dissemination free-riders who are cooperating

agents but they will create an isolated subnetwork which is completely separated from the network of the honest agents.

6.3.3. Malicious behavior

Next we look at two types of malicious behavior: forking and transaction hiding. Similar to the free-riding, the combination of our anti-entropy mechanism and the internal agents state transparency should isolate the malicious agents. The results of the experiments are presented in a similar format as previously in Figure 6.4.

As expected the agents are not able to reach any significant amount of transactions. The transaction hider is able to obtain one successful interaction, which is the one that is supposed to be hidden later on. The agent is not able to hide any transactions because any future partner expects the complete chain from that agent. Any missing blocks in the chain will be seen as a manipulation attempt.

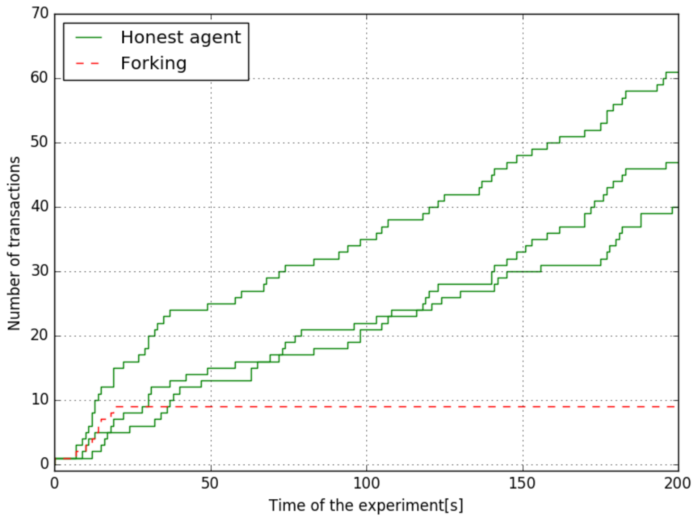
The forking agents creates a fork with a chance of 25%. The expected number of transactions before the fork happens is therefore 4. After the fork happens, the forking agent is detected by a partner once the partner receives the two conflicting blocks. Depending on the peer selection which is random, this can take some time. Therefore the forking agent is able to perform 9 interactions in the first 20 seconds of the experiment. After those interactions the red line of the forking agent is horizontal, meaning no more successful interactions happen.

6.3.4. Verification free-rider

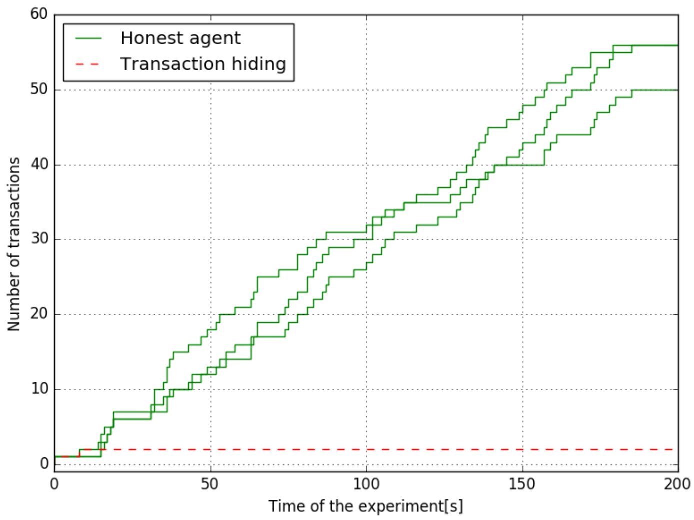
In the previous experiment we showed that malicious agents can be detected and isolated. In chapter WHERE EXACTLY we have presented the replay verification mechanism to show that not only malicious agents, but also those agents that knowingly interact with those malicious agents can be isolated. We first show how three honest agents act in the presence of one malicious agent and one verification free-rider. It is expected that the malicious agent should properly be ignored by the honest agents, but without replay verification, the verification free-rider interacts will all peers equally.

The results are presented in Figure 6.5. As expected, the verification free-rider, represented by the blue dotted line in the figure is able to perform just as well as the honest agents. The forking agent is mostly ignored. The interaction matrix in Figure 6.5b shows that most of the interaction of the forking agent come from the verification free-rider who does not perform any checks and therefore does not care about the fork.

Next we repeated the experiment, but this time the honest agents perform replay verification of their partners before every interaction. The results are shown in Figure 6.6.

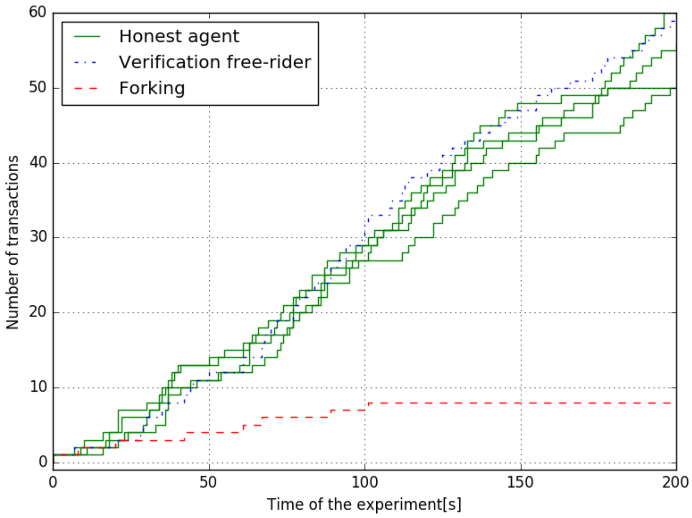


(a) Transaction history of three honest agents interacting with one strategic manipulator who performs a fork



(b) Transactions over time of three honest agents with one strategic manipulator who tries to hide a transaction

Figure 6.4: Experiments of honest agents with sinlge malicious agents

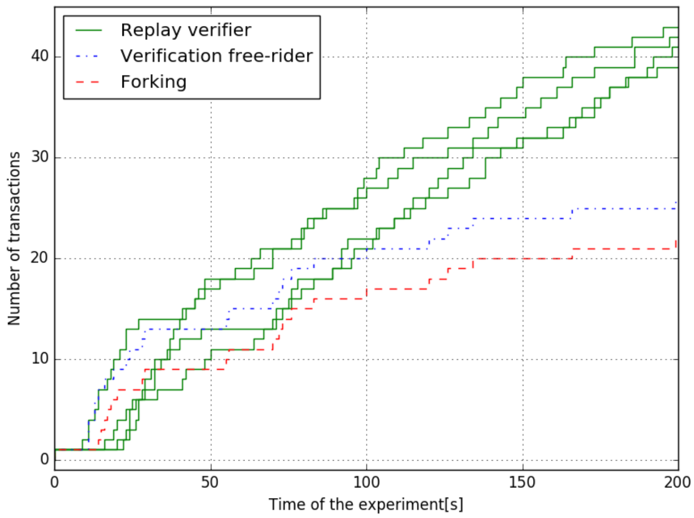


(a) Transaction history of three honest agents interacting with one strategic manipulator who performs a fork and a verification free-rider

(b) Interaction matrix of three honest agents with one strategic manipulator who performs a fork and a verification free-rider

Figure 6.5: Experiment with three honest agents without replay verification, one malicious agent and one verification free-rider

The results show that after a long initial phase in which the blue and red curve seem to follow the green curves, they do get shallower after around 100seconds. It is quite obvious, at least from the 100 second mark onwards, that the blue and red curve are running excatly parallel. This is because after detecting the forking agent and the verification free-rider, the honest agents are able to isolate the dishonest agents and ignore them for future interactions. Also the interaction matrix shows that in the end, both dishonest agents get few interaction with the honest agents.



(a) Transaction history of three honest agents with replay verification interacting with one strategic manipulator who performs a fork and a verification free-rider

(b) Interaction matrix of three honest agents with replay verification with one strategic manipulator who performs a fork and a verification free-rider

Figure 6.6: Experiment with three honest agents without replay verification, one malicious agent and one verification free-rider

7 Discussion

In this work we presented a strategy-proof mechanism for information dissemination. Applied to our distributed blockchain based trust system we are able to effectively defend against dissemination and verification free-riders. It creates an incentive for each agent on the network to help defend the network against any lazy or malicious behavior. It thereby is a major step towards a secure, distributed and scalable trust system.

* we defined a new blockchain system based on TrustChain which provides internal agent state transparency/gossip transparency * we formally proof that the architecture provides a complete view of the internal state of the agent * we defined a specific mechanism that makes use of the archiecture * we experimentally proof that honest agents are able to eventually identify free-riders and malicious agents

7.1. Future research

7.1.1. Further developing this mechanism

* incremental * research scalability properties for this mechanism * locality by interacting with those that have similar information * sybil attack resistance by checking that new agents paid their dues

7.1.2. Next steps for the trust system

* locality with ping *

Bibliography

[1] How volkswagen's "defeat devices" worked. https://www.nytimes.com/interactive/2015/business/international/vw-diesel-emissions-scandal-explained.html. Accessed: 2018-06-14.

[2] Facebook and cambridge analytica: What you need to know as fall-out widens. https://www.nytimes.com/2018/03/19/technology/facebook-cambridge-analytica-explained.html. Accessed: 2018-06-14.

[3] Karl Aberer, Philippe Cudré-Mauroux, Anwitaman Datta, Zoran Despotovic, Manfred Hauswirth, Magdalena Punceva, and Roman Schmidt. P-grid: a self-organizing structured p2p system. *ACM SIGMOD Record*, 32(3):29–33, 2003.

[4] George A. Akerlof. The market for "lemons": Quality uncertainty and the market mechanism. *The Quarterly Journal of Economics*, 84(3):488–500, 1970. ISSN 00335533, 15314650. URL http://www.jstor.org/stable/1879431.

[5] R Axelrod and WD Hamilton. The evolution of cooperation. *Science*, 211(4489):1390–1396, 1981. ISSN 0036-8075. doi: 10.1126/science.7466396. URL http://science.sciencemag.org/content/211/4489/1390.

[6] A.R.A.M. Chammah, A. Rapoport, A.M. Chammah, and C.J. Orwant. *Prisoner's Dilemma: A Study in Conflict and Cooperation*. Ann Arbor paperbacks. University of Michigan Press, 1965. ISBN 9780472061655. URL https://books.google.nl/books?id=yPtNnKjXaj4C.

[7] Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. Epidemic algorithms for replicated database maintenance. In *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, pages 1–12. ACM, 1987.

[8] Mark Granovetter. Economic action and social structure: The problem of embeddedness. *American journal of sociology*, 91(3):481–510, 1985.

[9] Sandra M Hedetniemi, Stephen T Hedetniemi, and Arthur L Liestman. A survey of gossiping and broadcasting in communication networks. *Networks*, 18(4):319–349, 1988.

[10] Ferry Hendrikx, Kris Bubendorfer, and Ryan Chard. Reputation systems: A survey and taxonomy. *Journal of Parallel and Distributed Computing*, 75:184 – 197, 2015. ISSN 0743-7315. doi: https://doi.org/10.1016/j.jpdc.2014.08.004. URL http://www.sciencedirect.com/science/article/pii/S0743731514001464.

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

[12] Sepandar D Kamvar, Mario T Schlosser, and Hector Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *Proceedings of the 12th international conference on World Wide Web*, pages 640–651. ACM, 2003.

[13] Zaki Malik and Athman Bouguettaya. Rateweb: Reputation assessment for trust establishment among web services. *The VLDB Journal—The International Journal on Very Large Data Bases*, 18(4):885–911, 2009.

[14] Michel Meulpolder, Johan A Pouwelse, Dick HJ Epema, and Henk J Sips. Bartercast: A practical approach to prevent lazy freeriding in p2p networks. In *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pages 1–8. IEEE, 2009.

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

[16] Martin A Nowak. Five rules for the evolution of cooperation. *science*, 314(5805):1560–1563, 2006.

[17] Pim Otte, Martijn de Vos, and Johan Pouwelse. Trustchain: A sybil-resistant scalable blockchain. *Future Generation Computer Systems*, 2017. ISSN 0167-739X. doi: https://doi.org/10.1016/j.future.2017.08.048. URL http://www.sciencedirect.com/science/article/pii/S0167739X17318988.

[18] Pim Otte, Martijn de Vos, and Johan Pouwelse. Trustchain: A sybil-resistant scalable blockchain. 09 2017.

[19] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

[20] Paul Resnick, Ko Kuwabara, Richard Zeckhauser, and Eric Friedman. Reputation systems. *Commun. ACM*, 43(12):45–48, December 2000. ISSN 0001-0782. doi: 10.1145/355112.355122. URL http://doi.acm.org/10.1145/355112.355122.