Creating trust through verification of interaction records

by

Jan-Gerrit Harms

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Friday August 31, 2018 at 4:00 PM.

Student number:	4163400	
Project duration:	January 16, 2018 – August 31, 2018	
Thesis committee:	Dr. ir. J. A. Pouwelse,	TU Delft, supervisor
	Prof. dr. ir. D. H. J. Epema,	TU Delft
	Dr. N. Yorke-Smith,	TU Delft

Abstract

Trust on the internet is largely facilitated by reputation systems on centralized online platforms. However reports of data breaches and privacy issues on such platforms are getting more frequent. We argue that only a decentralized trust system can enable a privacy-driven and fair future of the online economy. This requires a scalable system to record interactions and ensure the dissemination and consistency of records. We propose a mechanism that incentivizes agents to broadcast and verify each others interaction records. The underlying architecture is TrustChain, a pairwise ledger designed for scalable recording transactions. In TrustChain each node records their transactions on a personal ledger. We extend this ledger with the recording of block exchanges. By making past information exchanges transparent to other agents the knowledge state of each agent is public. This allows to discriminate based on the exchange behavior of agents. Also, it leads agents to verify potential partners as transactions with knowingly malicious users leads to proof-of-fraud. We formally analyze the recording of exchanges and show that free-riding nodes that do not exchange or verify can be detected. The results are confirmed with experiments on an open-source implementation that we provide.

Preface

The big hype around Bitcoin and other cryptocurrencies seems to create a distorted picture of the original idea of a distributed, democratic financial system. The general public does not appreciate the simplicity and brilliance of the concept but rather the price swings and the possibility to become rich.

A similar distortion happened to the internet itself. Once envisioned as an open and free network for information exchange, the World Wide Web has been transformed into an infrastructure for commerce, data collection and cyber attacks. Certainly, many companies have created solutions that simplify our lives significantly. But the free use of all kinds of services comes at an invisible cost. Slowly, that cost becomes more visible as targeted misinformation campaigns are dismantling democracies and data collection policies of the big internet monopolies are uncovered.

Decentralization is a natural solution to this problem. Some fundamental services such as transferring money, communication and the access to information should not be controlled by single entities and exploited for profit. In the same sense, creating trust with other internet users should be enabled through a decentralized system. This is the motivation to contribute to the design of a distributed trust system.

Working on this thesis has been a journey through many fields of science, from evolution theory and social sciences to synchronization methods of replicated databases. It has in many ways opened my eyes to the problems of centralized organizations and the advantages that distributed systems offer. Also, it sparked my interest in trust, cooperation and their relation to problems of society.

I would like to thank my supervisor Dr. Johan Pouwelse for his enthusiasm, infinitely large pool of ideas and suggestions as well as his invaluable feedback. I thank the Tribler team for keeping me awake with coffee and the review work of Quinten and Martijn. I thank my parents, my girlfriend, Lyla, and my closest friends for their support in moments of self-doubt. A special thanks goes to my housemates Marnix and Liesette who helped me through the last weeks. I would also like to thank Alexander for our interesting talks at lunch. Without you it wouldn't have been possible.

Jan-Gerrit Harms
Delft, August 2018

Contents

1 Introduction

Trust is the bedrock of society. From the evolution of species [? ? ?], economic markets [?] all the way to the modern sharing economy [?], trust impacts 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. These larger, wide-spread internet communities lead to more transactions of value between previously unrelated strangers. This makes a reliable trust creating system even more important. Protecting and distributing the knowledge about transactions, that these systems rely on, is a tough challenge we are faced with when designing a global trust system.

The sharing economy is a good example where trust needs to be built between strangers. The reputation systems of Uber[1] and Airbnb[2] are an essential part of their business. A 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 platforms, giving providers of those platforms great power and influence.

A similar situation exists in the financial 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. Instead of centralizing power, Bitcoin distributes the responsibility of verifying and confirming transactions over all users.

We argue that 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 chapter introduces some key concepts of 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 understand the problem description and the proposed solution in the following chapters.

1.1. Relevant 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 computer science [?]. In the simplified form of a model trust can well be described and studied. The Prisoner's Dilemma [?] is one such model from game theory that creates

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

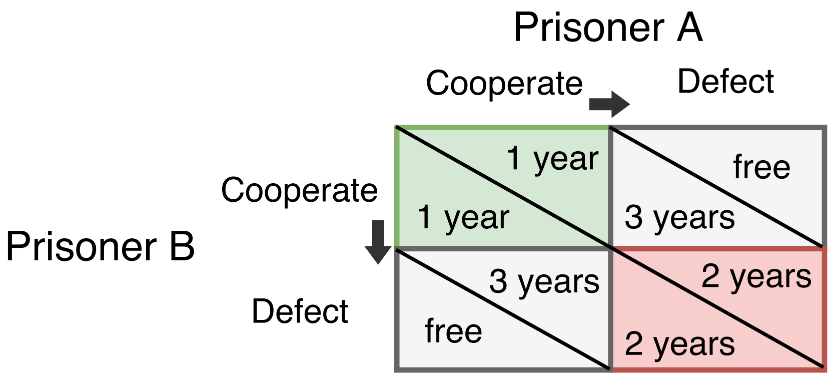


Figure 1.1: The sentences for two prisoners in the Prisoner's dilemma. The green outcome would be the best combined sentence of 2 years, while the red outcome is the worst. The arrows indicate that defecting is the dominant strategy.

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 Prisoner's Dilemma describes 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 avoid 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 [**?**]. In their work Axelrod and Hamilton [**?**] 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 again the Prisoner's Dilemma, but Axelrod and Hamilton let agents compete in multiple iterations of the dilemma. In the so-called Iterated Prisoner's Dilemma different strategies can be tested. 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. The best strategy was also the simplest one, called *tit-for-tat*. The agent would cooperate in the first round and afterwards imitate the partner's action.

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 [**?**] defined in total five forms in which coop-

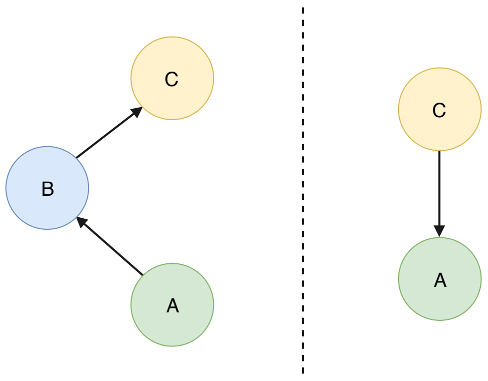


Figure 1.2: Indirect reciprocity means that A helps B which motivates B to help C. At some point C meets A and may return the favor to A, knowing that A has previously helped others.

eration 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.

1.1.3. Direct vs indirect reciprocity

Especially the concept of *indirect reciprocity* plays a major role in global-scale communities. The large size of such communities makes repeated interactions between the same partners, as in the Iterated Prisoner's Dilemma, less likely. Instead, most interactions will be with a new person. Also, in the Prisoner's Dilemma both agents can help each other which is also not the general case. Instead, often person A can help person B, yet B cannot be useful for A. However, B is able to contribute towards C. Figure **??** shows this flow of contributions. At some point C might meet A and be able to provide value.

Therefore, a variation of the Prisoner's Dilemma game has been studied by evolution theorists, which we will refer to as the *donation game*. In it players are either donor or recipient for one round, the donor can decide to either donate some value to the recipient or not. In subsequent rounds, players will switch roles such that they are donor and recipient in equal number of rounds. Donating comes at a small cost but largely benefits the recipient. In this game agents again can follow certain strategies, for example simple strategies like always donate or never donate. A more realistic strategy is, similar to tit-for-tat, donate to other that have also donated. If all agents donate on the first round, and knowledge of those donation is spread to other players, in the following rounds also all players will donate and cooperation is certain. Players who deny a donation will also be denied by others. Research has shown that if the costs compared to the benefit are sufficiently low and the history of behavior of partners is sufficiently well-known to players, cooperation can be established [**?**].

The stability of the cooperative strategy in the donation game depends on the knowledge about what strategy a partner adhered to in previous rounds. If A is has donated value to B and in a later round C has to decide whether to donate to A, knowledge of A's previous interaction influences the decision. If C knows about A's donation, C will most probably also donate, otherwise this might not be the case. This knowledge is generally known as *reputation*. Through donating to others agents build a reputation of being altruistic. If this reputation is spread, others can reward the altruistic reputation with a donation towards them. Spreading of reputations is essential to ensure cooperation and also happens in human society where it is known as *gossiping*. We talk about other people in order to estimate their reputation and build trust. A visualization of this process is depicted in Figure **??**.

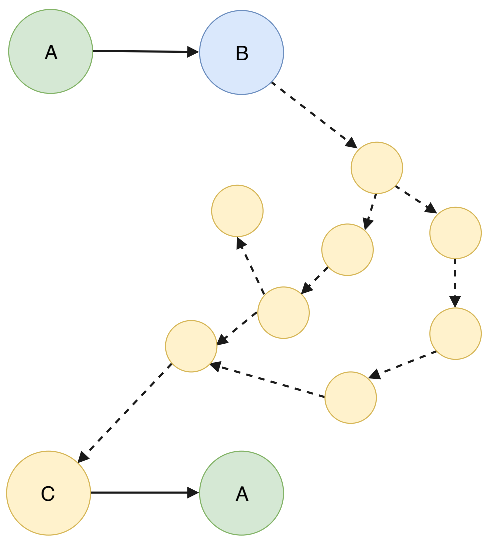


Figure 1.3: The knowledge of a donation from *A* to *B* is spread through a community, building *A*'s reputation. *C* hears about *A*'s reputation and donates to *A*.

1.1.4. Economics

The Prisoner's Dilemma, trust and cooperation are not only studied by evolution scientists but also economists. A marketplace is a good example for a context in which strategies for indirect reciprocity can be applied. How trust and the lack of it influence trade and markets has been studied in the well-known paper "Market for lemons" by Akerlof [**?**]. Akerlof describes in his paper how the information asymmetry between buyers and sellers on the market for used cars can lead to a general decrease in the quality of cars sold in the market. Buyers cannot distinguish between good and bad used cars and therefore offer an average price. However sellers will only sell bad cars for the average price in order to make a profit. This leads to a general decrease in quality which buyers will observe and subsequently offer less. Akerlof describes institutions to solve this problem 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. [**?**]

1.1.5. Trust and reputation

We have used the terms trust system and reputation system interchangeably so far. We shall shortly shed light on the differences.

Commonly a system is referred to as a reputation system if it produces a score for an entity as seen by the whole community where as trust is the subjective view of that reputation. If the whole history of an agent is known, only a single reputation score can be assigned to that agent. Yet, each observing agent can have a different trust score for the agent given the same evidence [**?**].

However, reputation can be subjective if each agent only knows a subset of an agent's history. In that case each different subset creates a different (subjective) reputation. Both reputation and trust system thus rely on the knowledge of an agent's history. The difference lies in the calculation of scores.

This work will mostly be concerned with the way the history of agents is communicated through records of interactions. As this is significant to both trust and reputation systems, we will continue to use them interchangeably. However, we are aware of the existing difference.

1.2. Evolution of trust systems

Although the role of trust for marketplaces is well described by the market for lemons, the ways of building trust have evolved as shown in Figure **??**. 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. Trust was mostly based on personal experience or word-of-mouth marketing.

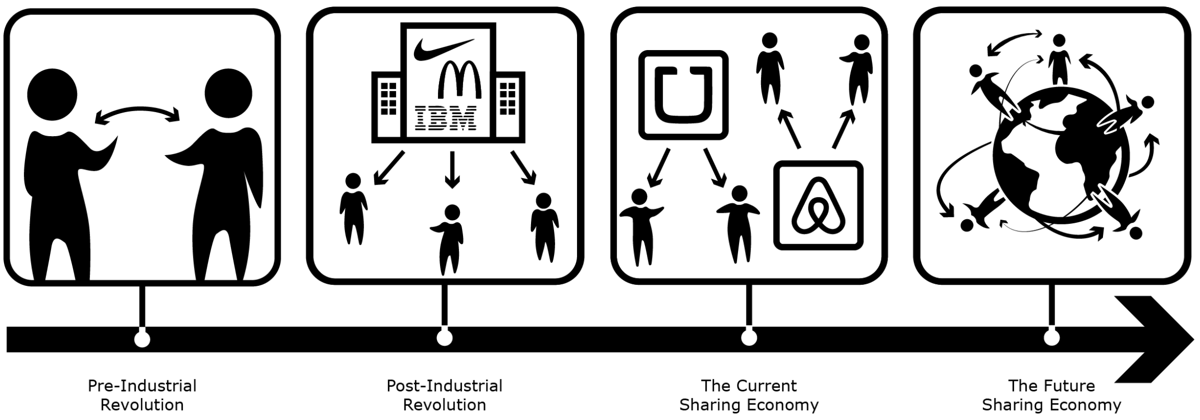

IBM
Pre-Industrial Revolution
Post-Industrial Revolution
The Current Sharing Economy
The Future Sharing Economy

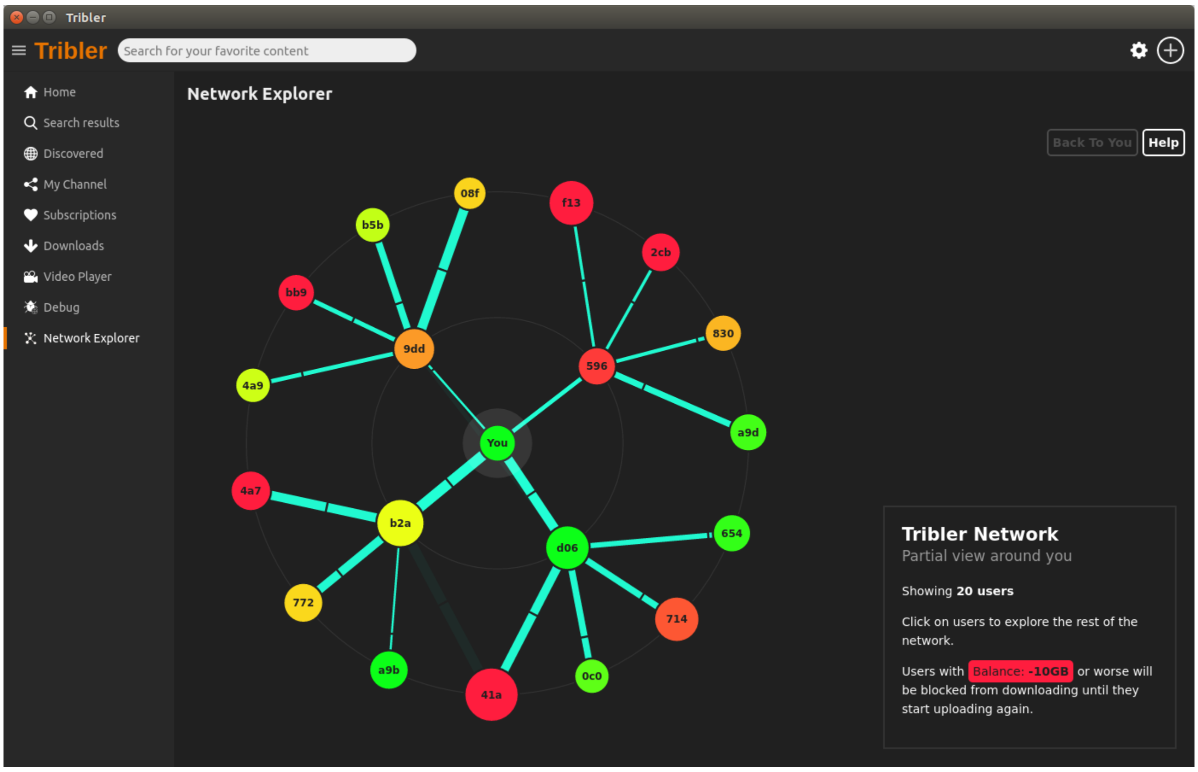
Figure 1.4: The evolution of economies. Trust played a role in each phase. First trust was established between traders. Globalization led to trust between large companies and their customers through branding. The current sharing economy creates trust between strangers through a centralized platform. We envision a future in which trust can be created directly between strangers using an open, free and independent distributed trust system. Source: Created at Blockchain Lab TU Delft

During the industrial and post-industrial age, companies have largely replaced local producers and are trusted by millions of customers based on their brand name. We buy most goods from nation wide and global companies that are competing with few other competitors. Trust is build through advertisements and product quality but also the effect of masses plays a role: if many people have an iPhone, even more people want one. But the trust has limits. A recent example of a company misusing the trust in the brand is Volkswagen [**?**], who advertised and sold dirty Diesel cars under the cover of being clean.

The last two decades saw the development of a new branch of the economy: the sharing economy built on the internet. We can now buy used goods on eBay, rent a house to a stranger (or from a stranger) on Airbnb, get into a stranger's car with Uber. The sharing economy or collaborative consumption is the rising star of economic concepts in the information age and it is powered 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. Similar to what can be seen in the analog economy, digital companies first build trust with their users before abusing their power. Again a recent example is the Facebook/Cambridge Analytica scandal [**?**].

Similar to the above examples of a misuse of trust, customers of banks lost trust in them when large scale speculations in the housing market led to the 2008 financial crisis [**?**]. Shortly afterwards an alternative financial system was proposed with Bitcoin [**?**]. Bitcoin enables digital money transfers without banks, so direct transactions between owners of bitcoins in a decentralized system. The advent of Bitcoin represents the next step in the digital economy, in which market interactions happen directly between entities without an intermediary.

We envision a future in which collaborative consumption is possible through direct trustful connections. This future requires a trust system which is application agnostic, owned by no one and ruled by everyone: a distributed 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. Mechanisms need to be designed to encourage honest behavior in the system. Although Bitcoin has proven that secure interactions in a distributed network are possible it struggles with major scalability issues [**?**]. Distributed, secure and globally scalable systems remain an unsolved problem.



Tribler Network
Partial view around you

Showing **20 users**

Click on users to explore the rest of the network.

Users with Balance: -10GB or worse will be blocked from downloading until they start uploading again.

Figure 1.5: Graphical user interface of Tribler. The network graph shows which nodes we have interacted with in the past. The stroke width of the connections shows their reputation from our point of view.

1.3. Tribler and its relation to trust systems

Distributed trust and reputation systems are not completely new. Examples of reputation systems being applied in a distributed application can be found in the field of peer-to-peer data exchanges and mobile ad hoc networks [**?**]. In both contexts a reputation system can ensure the fair use and contribution to the network's pool of resources. In this section we introduce Tribler, a peer-to-peer file-sharing and video streaming service which is based on the BitTorrent protocol. Taking Tribler as an example, we show how an application context can be mapped to the dilemma and trust problem which were discussed so far.

1.3.1. Trust and reputation

The uploading and downloading of data in Tribler is a social dilemma as each user wants to stream videos while contributing as little as possible. The peer-to-peer video streaming use case can be mapped to the donation game we described in Section **??** very well. When a node streams a video, it downloads the video data from another node which has that content available. The other node needs to upload the video to the downloader. The uploader matches the donor, while the downloader is the recipient. The uploader does not immediately benefit from uploading a video. He only benefits if the knowledge about the altruistic act is spread such that he is more likely to receive videos in the future.

The reputation of an agent in Tribler then represents the contribution behavior of that agent. Contributing, that is uploading video streams to other users, increases an agent's reputation whereas downloading, that is streaming videos, decreases an agent's reputation. In contrast to reputation, trust is subjective and can be seen as the subjective perception of a peer being a "good contributor".

In order to better illustrate the difference between reputation and trust we provide an example. Assume an agent A knows two agents, B and C and sees that both have uploaded and downloaded the same amount of data. However, B has uploaded most to A while C has uploaded only to agents that A has not heard of. Human intuition tells us that A should trust B more than C, because of the stronger direct connection. Trust therefore depends on the topology of the network.

1.3.2. Agents

In the iterated Prisoner's Dilemma, agents are the entities that take part in games. Intuitively one would assume that users are the agents in Tribler. It would be more accurate though to say that a running instance of the Tribler software is the agent. A single human user can have multiple instances of the software running on multiple machines and thus "control" multiple agents. Despite being able to run or stop the agents, the human user has only a very small decision space, though, because the software runs on its own and executes according to design.

The distinction of *honest*, cooperating and *dishonest*, defecting agents is then made as follows: honest agents are the instances that run the software as intended while dishonest agents are instances of manipulated software.

In the following description of the system we will use both, "node" and "agent" when referring to the behavior of a Tribler instance executing some action.

1.3.3. Building and spreading reputations

As explained in Section **??** building and spreading the reputation is important for cooperation to be possible. A distributed system does not allow for a central storage or communication tool to track and disseminate reputations. Instead, a solution similar to the human-to-human gossip is necessary to spread knowledge of previous transactions. Similar to humans, Tribler instances can communicate with each other, thus spreading knowledge of contribution behavior seems to be straightforward.

We argue that the opposite is true. As the reputation bears value, agents can try to misreport transactions in order to generate a better reputation than when being honest. Agents can also create fake reports of transactions with non-existent agents or distribute conflicting reports. We need to explore how transactions should be recorded and disseminated in order to ensure that bad reports and dishonest agents can be identified and dealt with.

1.4. Our contributions

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. We show that current approaches are either not scalable enough or cannot guarantee the detection of manipulation attempts. We claim that a scalable solution requires proper incentives which encourage each agent on the network to help defend the system through sharing and verifying the records of transactions. Specifically this work extends our TrustChain architecture with a new type of record: exchange blocks. Together with transaction blocks these enable our system to document all communication between agents. This enables the honest agents to verify that their peers follow the proper exchange and verification policies. We further implement this architecture and show that if honest agents follow a policy to exchange all data, they are able to detect and ignore free-riders, strategic manipulators and colluding agents, who do not verify their partners.

We will enlarge on the problem description in the next chapter. Afterwards TrustChain is introduced and explained. In Chapter **??** we theoretically study the importance of gossiping and ways to prevent any free-riding on the security offered through peers. Next, we define our extension to the TrustChain fabric and propose a specific mechanism of exchanging data. Finally, we confirm the ability of defending against manipulation and free-riding in experimental analysis. The final chapter concludes the work and makes suggestions for further research.

2 Problem description

Our audacious long-term ambition is to create a global, distributed trust system. Such a trust system as a public, free and non-profit utility is a key element to enable next-generation online applications. In the previous chapter we have introduced trust research and the influential role trust plays in human society. The design complexities of such a system reach far beyond the scope of a single master thesis. In this chapter a specific problem will be defined that we identified as a crucial challenge to solve in order to achieve the long-term goal.

2.1. Model of trust and reputation

Trust comes natural to people. Our intuition tells us from experience or some evidence who is trustworthy and who is not. This is not a failsafe process but it is the best we can do. If we try to reason about this vague concept of trust or try to implement trust into an automated agent we cannot use that intuition. Instead a model is needed which describes in explicit terms the concepts, and the relationships between them. This helps us to afterwards determine the components that are needed to create a fully capable trust system.

"A Computational Model of Trust and Reputation" by Mui et al. [**?**] is an important early work which provides a complete computational basis for those vague concepts. Mui defines the concepts of trust and reputation in the context of an embedded social network. The network consists of a set of uniquely identifiable *agents* that are able to interact with each other. Similar to the Prisoner's Dilemma agents either cooperate or defect in the *encounter*. The incentive for an agent to cooperate in encounters is the hope that the same agent or another agent will return the favor and also cooperate in future encounters. Throughout their lifetime agents build a *history* of encounters. The reputation of an agent is determined by that history.

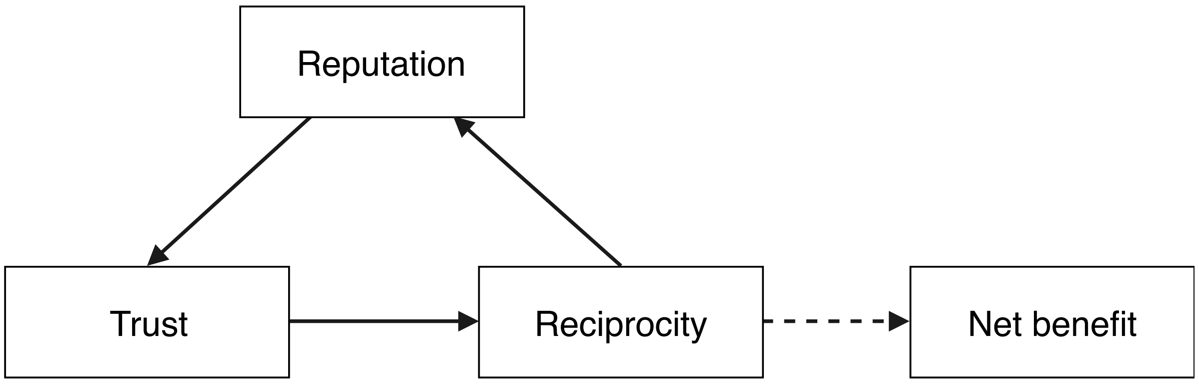


Figure 2.1: Self-reinforcing relation between trust, reputation and reciprocity. Source: Mui, 2002 [**?**]

Trust and reputation can be calculated. For example, an estimator for the reputation of an agent can be the number of encounters in which the agent reciprocated over the total number of encounters. Trust, is the probability that an agent will reciprocate in the next encounter given the agent's history. Mui et al. defines reputation as " perception that an agent creates through past actions about its intentions and norms". A good reputation leads to trust from other agents. Their definition of trust as “a subjective expectation an agent has about another’s future behavior based on the history of their encounters” reflects this. The more trust an agent has in another, the higher the probability for reciprocation in the following encounter. This leads to another encounter of reciprocation, effectively increasing the agents reputations. This circular relationship is represented in Figure **??**.

This self-reinforcing system of trust and reciprocity benefits all agents in the system. If agents follow strong reciprocity norms, cooperation, is the optimum for the system as a whole, can be ensured. Defining a computational model allows to automate this process of selecting trusted partners and enables global communities of both human and automated agents to have trusted connections.

2.2. Trust system architecture

The concepts defined in the above model allow people as well as automated processes to make trustful decisions. It also defines, explicitly and implicitly, the main components which are required to create trust between agents. We can use this conceptual description to create an architecture for our proposed trust system. Making a trust system distributed poses another major challenge. We shall briefly describe each of the components and their challenges. Their relation is visualized in Figure **??**.

1. **Identity Layer.** At the lowest level there is the identity layer that ensures an agent is identifiable and distinguishable from other agents 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 fake identities and spamming are possible. 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. A long-term goal is to create identities which are self-sovereign, i.e. identities built, managed and used by the owner without central authority [**?**].

2. **Communication Layer.** In order to interact, agents need to be able to connect and communicate. The internet creates a global communication network with high connectivity across the world but it is currently not in a state that point-to-point 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(NAT) creates subnetworks with local address spaces. Connecting from such a subspace to a server with a public address is still simple, but point-to-point communcation, 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. Our research group has designed and implemented a communication layer which solves most of these problems. The system, called IPv8, is open-source and published on GitHub[1].

3. **Encounter Record Layer.** Reputation is based on the history of encounters. This history needs to be recorded and distributed such that agents can lookup and reason about each others history. Also the integrity and correctness of records needs to be ensured. 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.

4. **Interpretation Layer.** 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 explored different reputation functions like ones based on maximum flow calculations(NetFlow [**?**], MaxFlow [**?**]) and random walks(PageRank [**?**]).

5. **Application Context.** We imagine that the trust system we are developing can be used for any type of application that requires two entities to trust each other. The access to the trust system should be without any cost and open to anyone who is able to connect to the internet. The exact implementation of the Encounter Record Layer as well as how to interpret those records in the Interpretation Layer is dependent on this application context.

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

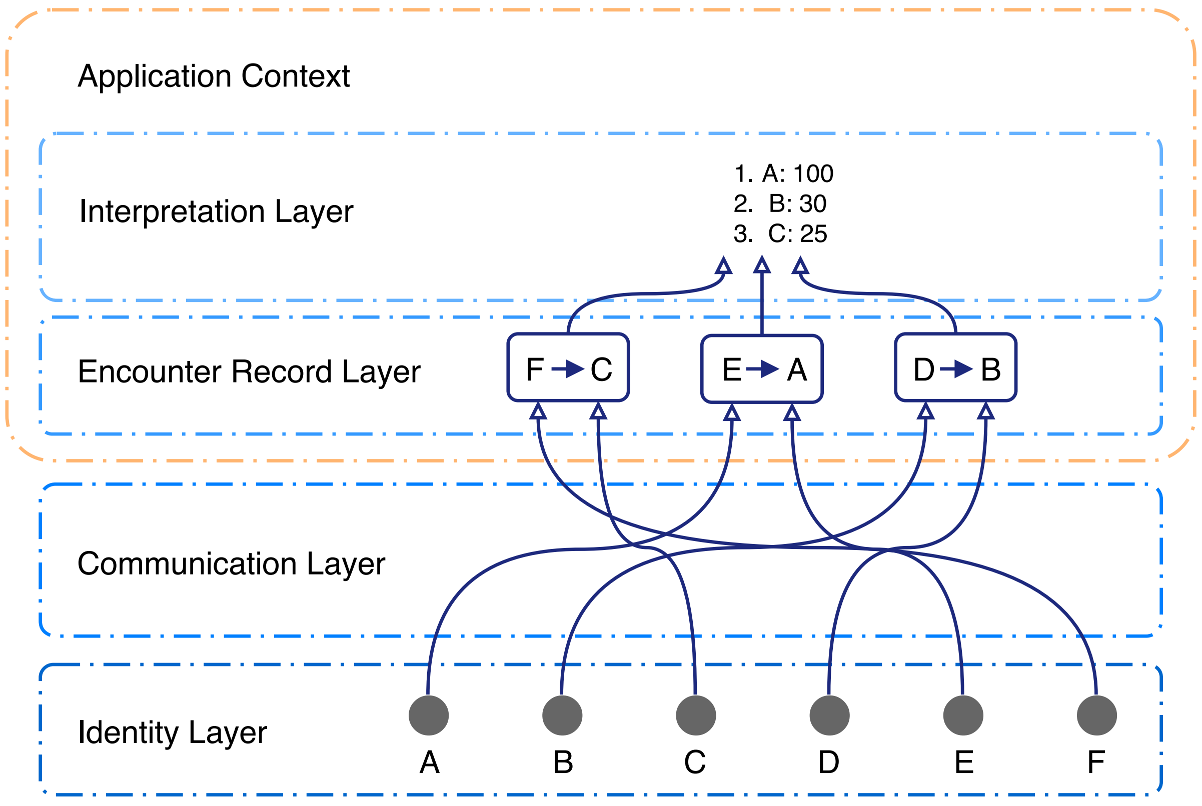


Figure 2.2: Layered structure of our trust system architecture. The Identity Layer ensures the authenticity. The Communication Layer allows for point-to-point connections. The Encounter Record Layer ensures the correctness of records. The Interpretation Layers allows to calculate reputation and trust of peers. Both the encounters and their interpretation happen within a certain Application Context.

Each layer adds another level of protection: if identities are expensive and hard to create, fake identities will be less likely, protecting the system against spamming. Creating an immutable record of each interaction and distributing that information to everyone makes agent's histories tamper-proof, unchangeable and 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.

Each of these components poses major challenges to the designers of a trust system. Yet we find that especially the recording, distribution and verification of encounters is crucial to the success of the trust system. Records of encounters are at the basis of the calculation of reputation and trust. Thus, any open challenge in this layer will propagate up into the layer of reputation, trust and the application context.

2.3. Manipulation resistant interaction histories

We now describe in more detail the challenges of ensuring the correctness of interaction records.

First, we should discuss the value of reputation and trust. In order to build a good reputation, people act pro-socially at a personal cost which is rewarded with other people 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.

Agents base their perceived reputations of other agents on the knowledge about past interactions of their peers. In a centralized system all user's actions are observable by the single central entity. But this is generally not true for distributed systems as interactions are generally hard to to observe. On the other hand, using only one's own history of interactions as indicator for our peers reputation is also not an option. In a trust system with millions of users chances are high that users interact with many previously unknown people. Instead agents need to acquire knowledge about their peers transactions to make better decisions and reward those with a good reputation. This is possible with epidemic gossiping: agents report new interactions to other agents who further spread those news. This is a natural process that also happens in the real-world trust building process [**?**].

Two main challenges of this process can be identified. Firstly, how can we ensure that agents are honest about their own interactions? We have established that reputation has a value to agents. That could lead to agents reporting a manipulated history which is better than the true history in order to obtain that value for free. Agents can protect against misreporting by exchanging information with each other and verifying that they "heard the same" about another or others. This leads to the second challenge: how can we ensure that agents take part in this process of exchanging and verifying information? Although each honest agent benefits from banning dishonest agents from the network, an agent can still argue to only report an agent that directly defrauded him. Checking any other behavior of the partner does not directly benefit him. It is therefore a social dilemma: if all agents help with verifying each other, everyone will benefit. Yet a single verifier has only little influence.

2.4. Research question

With the previous discussion about the challenges of reporting interactions, we can define a scope for this work. We are aiming to facilitate honest reporting in a global trust system in order to create a basis for valid reputation and trust calculations. The challenge of this system is to ensure the integrity of these reports.

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

How to ensure that reports of encounters are honest, consistent and disseminated in a distributed trust system?

In our research question we state three desireable properties of encounter reports that our system should enforce.

- **Honest.** A record of an encounter includes an action performed by both agents. In the simplest case, as in the model we introduced to the reader in this chapter, this is simply cooperation or defection. The actions need to be honest in order to rely on them for trust calculations. Similarly the order of encounters and the participating parties need to be honestly reported.

- **Consistent.** We need to make sure that only one version of each encounter can exist. Multiple versions of the same encounter leads to ambiguity of agents histories and thus invalid trust calculations.

- **Disseminated.** Reputation can only be valuable if it is spread through the network. Instead of spreading reputation directly, we require to spread the knowledge of encounters such that each agent is able to determine their peers' reputation and whether to trust them or not.

Our reporting systems is supposed to be used as the basis for a global scale, distributed trust system. From this description we can extract specific requirements for the reporting system.

- **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.

- **Manipulation-resistant.** Once reputation increases in worth users of the system will try to exploit it through attacks, alone or by colluding

Next to the properties defined in the research question, these practical requirements set boundaries for the solution that we can propose.

2.5. Related work

Before defining our own solution to the chosen problem we explore existing approaches.

2.5.1. Replicated state machines

The problem that we described in this section can be generalized to the problem of replicated state machines. The state of a reputation system depends on the interactions between agents. Each new encounter changes this state. The true state can be described by evaluating all records of encounters. Each agent in the trust system stores a subset of these records . Based on the records agents are able to approximate the global system state. If all agents have the exact same records, they are able to agree on the state of the system.

Replicated databases are a practical application in which this problem also arises. Early work from Demers et al.[**?**] introduces the anti-entropy mechanism 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. Anti-entropy is a process in which each database periodically chooses a random other instance and both exchange all database contents in order to resolve any differences between the two. In this work we will study a similar mechanism for exchanging records of interactions between agents.

Later similar processes of epidemic information spreading were summarized under the label of *gossiping*. In distributed systems randomly chosen pairs of agents communicate to share knowledge. Both agents' knowledge base is combined and both agents will further spread the combined information set in their following encounters. Note that this is a scientifically well-defined concept. different from the previously mentioned gossip in communities to facilitate indirect reciprocity. We will subsequently refer to this well-defined concept when mentioning gossip [**?**].

Very recent work combines the anti-entropy approach with a hashchain solution. Baird proposes a byzantine fault tolerant consensus scheme for replicated state machines [**?**]. Similar to the anti-entropy mechanism nodes in the network randomly connect and exchange all their knowledge. In each such encounter new transactions can be announced. By signing each encounter and connecting it to the previous encounters using their hashes, a hash graph is created which records the communication between all agents. Peers gossip about gossip and thus know what each of their peers have heard of and know. A byzantine fault tolerant consensus mechanism is defined based on the hashgraph.

In this work we make use of two core concepts described in this section. Firstly, the anti-entropy mechanism will be studied as an example for synchronizing the states of agents. This allows for agreement on reputations and trust, as well as fast dissemination of encounters with few necessary connections. Also, we will define an architecture for recording those exchanges, similar to Hashgraph. On the other hand, our architecture allows also for different exchange mechanisms that require less bandwidth than anti-entropy. Also agents do not need to actively gossip when not interacting: instead we propose to exchange information prior to any interaction.

2.5.2. Blockchain systems with global consensus

Replicated state machines were the predecessors of Bitcoin [**?**] which created a whole new interest in distributed systems by releasing a concept and implementation for digital payments without banks. Bitcoin's innovation is mostly a combination of multiple existing concepts into a (new) technology, called blockchain.

Although the application context is a different one, the problem of recording transactions between automated agents is very similar to our problem. Bitcoin was adopted quickly for anonymous digital payments. But its adoption layed bare scalability problems in the architecture. The proof-of-work algorithm increases its complexity with more active nodes and keeps the transaction throughput constant. This leads to a theoretical bound of 60 transactions per second [**?**]. Other similar concepts were proposed with Litecoin [2], Monero [3] and Dashcoin [4] which add minor improvements over Bitcoin. Ethereum [**?**] was the first blockchain based fabric that allowed the decentralized execution of scripts in addition to simple transactions. However it struggles with similar scalability issues as Bitcoin.

In current development are proposals for increasing transaction throughput in blockchain-based systems, through so-called layer-2 solutions. One proposal introduces off-chain transactions in a protocol called the Lightning network [**?**]. Two nodes deposit some Bitcoins into a channel. Afterwards they can use the channel for any number of transactions. Once they agree to stop transacting, a net settlement is published. Only the opening and closing of channels leads to transactions on the blockchain, all transactions on the channel itself is only recorded locally. Multiple hops through channels are also possible such that if A has a channel with B and B has a channel with C, a can perform a transaction with C. Although the lightning network improves scalability by allowing for an unlimited number of off-chain transactions, locking funds

[2]https://litecoin.org
[3]https://getmonero.org
[4]http://dashcoin.info/

in each open channel can be problematic. Also, if channel creation and maintenance is expensive, a natural result is that certain nodes keep channels with many other nodes to act as an off-chain hub. This leads to centralization.

Another development is sharding. By splitting the network into several sub-networks, called shards. Each shard records and secures its own state. As the network grows, new shards can be added, thus increasing the overall throughput of the system with increasing size. Although this seems like a good solution, problems arise when transaction need to be performed between shards. Elastico [**?**] uses a sharding solution with a Byzantine consensus scheme and allows for almost linear scalabilty. However, inter-shard transactions are not possible.

Other innovations focus more on the consensus algorithm itself. Proof-of-work is not only slow but also very energy-intensive [**?**]. Therefore other consensus protocols have been proposed and implemented such as Proof-of-State (POS) and Delegated-POS [**? ?**]. In the POS consensus mechanism, nodes do not need to expend computational resources to create a total sequence of blocks but rather bet some of their resources (stake) on a new block being the next on the chain. Betting on the wrong block or a faulty block will lead to loss of those resources. The stake to participate as a validator can be high, leading to centralization of power at the largest stakeholders. Delegated-POS allows each node to vote on a set of validator. The decentralization is re-established but by decreasing the parties involved in the validation of new blocks, the mechanism is not as secure as Proof-of-Work.

2.5.3. Pairwise accounting and pairwise ledgers

In parallel to the blockchain development a different variation of replicated state machines was developed in the form of pairwise accounting systems. Instead of announcing transactions to all nodes in the network and ensuring global consensus, in pairwise accounting systems interactions are initially only observed by the participating agents. Through employing some dissemination mechanims they are broadcast to other agents. With respect to the previously introduced systems which enforce global consensus, this solution makes for a scalable, though less secure, transaction record. However, there is no guarantee on which information agents are acting and consistency cannot be ensured.

Meulpolder et al. propose BarterCast, a system that records transactions in the BitTorrent network and selects future peers by their perceived reputation. The reputation is based on maxflow calculation of the interaction graph with agents being nodes and past transactions being edges. The goal was to detect and ignore lazy free-riders who do not upload in the BitTorrent network [**?**]. In later work other mechanisms were proposed to be more resistant to dishonest reporting and sybil attacks [**? ?**].

However the BarterCast system did not have any tamper-proof recording so the accounting mechanisms were vulnerable to manipulations. Therefore, recent work by Otte et al. proposes a blockchain-based, pairwise ledger which provides a tamper-evident scalable ledger. Transactions are entangled with two hash pointers to previous transactions. A transaction is confirmed by two parties without requiring consensus with the rest of the network. We will describe this solution in detail in Chapter **??** because it will be the basis for the extension this work introduces.

Similar to TrustChain, IOTA is a distributed ledgers without global consensus [**?**]. Transactions are initially only recorded by the participating agents. A proof-of-work algorithm is used to work against spamming of transactions but not to obtain global consensus. Each transactions is related to two other transactions that are confirmed. This creates a directed acyclic graph (DAG). A graph structure can grow much faster than a single global chain. DAG based ledgers therefore solve the largest problem of blockchain solutions: scalability.

Although DAG ledgers are superior to blockchain solution in terms of scalability, their weakness is consistency. Nodes are not guaranteed to act on the same information about transactions. This can lead to conflicts about the validity of transactions. It is key to incentivize agents to acquire information from other agents and verify any new transaction against their knowledge. In this work we propose a mechanism that guarantees this behavior.

2.5.4. Reputation systems

In our introduction we have mentioned the reputation systems of Airbnb and Uber as examples of state-of-the-art commercial systems for creating trust on the internet. Other such systems like eBay's reputation system for auctions have been more in the focus of research. It was found in multiple studies that a good reputation positively influences future interactions [**? ? ?**].

[**?**] is a very extensive survey of trust and reputation systems. The authors discuss among other things different definitions of trust and reputation, centralized and decentralized architectures of reputation systems as well as examples of commercial and live reputation systems. Apart from eBay, commercial reputation systems can be found in expert sites like AskMe[5], sites that offer product reviews like Amazon[6], discussion fora, for example Slashdot[7], Google's[8] page ranking which uses the PageRank algorithm[**?**] and Scientometrics. [**?**] is a more recent survey of reputation systems. They defined a 14 dimensional taxonomy to compare reputation systems, among them governance which is either distributed or centralized.

All commercial systems use a centralized architecture. Centralized architectures record interaction feedback on servers who handle all communications with users. A central server can guarantee the security and integrity of records of interactions as it a single source of truth. As discussed in Chapter **??** we do not see a central solution as a viable option.

[**?**] also mentions many academic reputation systems, a majority of those targets the decentralized governance scheme, mostly in peer-to-peer file sharing applications, such as EigenTrust [**?**], PGrid [**?**] and XRep [**?**].Although reputation system are much related to our use case, most work has been done on the the calculation of reputation rather than the secure storage of records. Also, they mostly have a conceptual description of the reputation system rather than a software implementation that is proven to work.

2.5.5. Other work

In [**?**] the system PeerReview is proposed which records in a tamper-proof hashchain the messages that each agent sends and receives. This allows each agent to verify that other agents act according to a reference implementation. In our work we use a similar approach but optimize the system for the recording of two party transactions instead of a general messaging scheme. This allows for a simpler witness selection process: each future transaction partner is a witness and failing to perform the verification will also mark witnesses as a fraud.

[5]http://www.askmecorp.com
[6]https://www.amazon.com
[7]https://slashdot.org
[8]https://google.com

3 Scalable blockchain accounting

Trust and reputation systems rely heavily on interaction records. We have shown that the problem of recording transactions in distributed system has been in the focus of research for many years. The commercial interest has lately spawned many blockchain solutions. In previous work our research group has deployed a blockchain-based recording system for interactions, TrustChain. This system is implemented in a peer-to-peer video streaming service Tribler and the IPv8 project[1].

This chapter introduces the blockchain and the TrustChain architecture itself to the reader. We argue that the TrustChain is a more suitable system for a trust system than a single global ledger. Also we analyze how the TrustChain architecture can be attacked. We point out that although detection of attacks is possible the system does not incentivize agents to share and verify data which is essential for the detection. In the following chapter we analyze this in the context of a model and propose an extension afterwards.

3.1. Blockchain basics

The design of distributed databases bears many challenges. Especially if sensitive data is involved and users need to have access globally, the asynchrony of events, lack of guarantees on data consistency and agent honesty create issues. For the early years of the internet those challenges seemed insurmountable. That is why most services that act on sensitive data are centralized, examples being banks, government institutions or commercial services like Facebook[2]. Centralization has its own shortcomings such as abuse of power, dishonesty, single point-of-failure and platform lock-in. We described those issues in more detail in Chapter **??**. The increasing significance of those issues led to the design and implementation of the Bitcoin protocol [**?**] for digital money transfers. The Bitcoin protocol allows a distributed network of agents to agree on the exact order of events, through a hash chain which acts as a distributed timestamp server and the proof-of-work algorithm. This architecture is commonly referred to as blockchain or distributed ledgers.

3.1.1. Concept

A blockchain is essentially an append-only database in the form of chained blocks. It is designed to be used as a secure information storage in distributed systems without any central governance.

Each block on the chain contains a set of transactions, the root hash of that set, a nonce (used in the consensus algorithm) and the hash of the previous block. By including a hash from the previous block, blocks are chained together as can be seen in Figure **??**. Any change to a block on the chain will change the hash of that block. This voids the following block's hash pointer because it points to the old block.

The first block in the chain is called the genesis block. The Bitcoin blockchain and similar single-chain ledgers only have one genesis block. Each block can also be identified by a sequence number $s \in \mathbb{Z}^{\geq}$, which is an increasing integer that starts from 0 at the genesis block.

All agents are acting on the same copy of the blockchain. The one global chain therefore contains all transactions that happen on the network. Single global ledgers create an incentive for agents to publish new blocks by rewarding them with currency. New blocks are accepted if they are on top of the longest chain. This

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

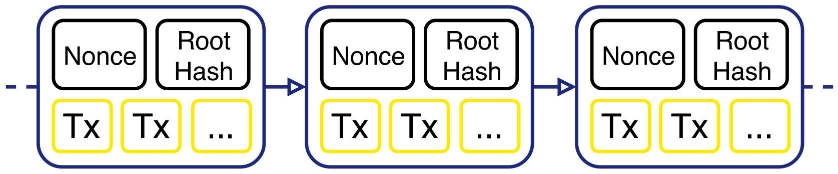


Figure 3.1: Conceptual depiction of a slice of a Bitcoin-like blockchain. Blocks contain multiple transactions and are chained together through the hash of the previous block. Source: *Creation of TU Delft BLockchain Lab*

creates an incentive for all participating miners to stay up to date on the state of the global ledger. The only way to create the next block and receive the block reward is acquire the latest state of the ledger.

3.1.2. Consensus

Transactions and new blocks need to be published on the network. The block creation process is managed by the consensus mechanism *proof-of-work*. This process is also called mining and the nodes that take part in it are thus miners. It works as follows: Each new block contains a computational intensive puzzle which needs to be solved in order to be able to publish a block. Specifically, miners are increasing the nonce x until $\mathcal{H}(x) < d$ where $\mathcal{H}$ is any secure hash function and d is a target hash. The smaller d, the more hashes need to be calculated before finding a hash. All miners are computing hashes until they find a correct x.

Once they find a possible solution, the agent combines all the transactions that were published since the last block and combines them to a new block. The block size is fixed such that after adding a certain number of transactions, the block is considered complete. If complete, the agent broadcasts the block in the network. Any agent that receives the block will verify that the solution to the puzzle is correct and all transactions are valid. If so, they add the new block to their copy of the chain and start with solving the next puzzle in order to become the next creator of a block.

The creator of the block is rewarded with transaction fees paid by the authors of the included transactions and a coinbase transaction. The latter is the generation of new coins from a finite supply of new coins. Over time, the size of the coinbase transaction reduces until it reaches 0. At that point, miners are only rewarded with the transaction fees.

Conflicts can arise when two blocks are found at approximately the same time. Each block will be considered the latest by part of the network. The network is thus partitioned and a natural fork is created in the chain. The conflict is resolved when the following blocks are mined. The partition with the larger CPU power will mine consecutive blocks faster. The longer chain will persist as the other partition switches towards it. The mechanism ensures that (except for the latest blocks) agents are acting on the same view of the network.

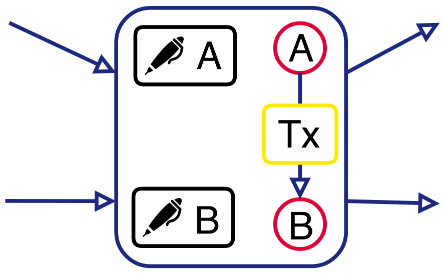
3.1.3. Tamper-resistance

The chain of hashes that connects blocks on the blockchain creates a theoretically tamper-proof history. Changes to any block will alter its hash and break the chain of hashes. Therefore any change to the block requires the recalculation of all following blocks. Because all blocks need to be agreed upon on the network other agents will be able to see such tampering with the blocks and will not agree on it.

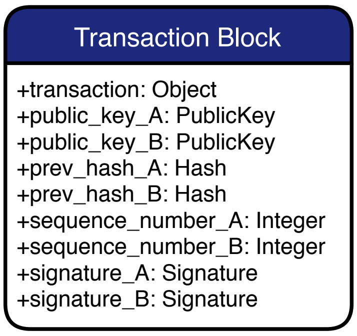
Similarly the order of blocks is ensured through the hash pointers. Any change to the order of blocks will will result in a break in the hash chain. The sequence of blocks also orders the transactions stored in the blocks. Any transactions in an earlier block happen before the transactions in the later block. Also, as was just explained, any change to the ordering of transactions. Even though this order does not necessarily correspond to the local time of the agent who published the transaction, the blockchain will create a globally accepted order for transactions. This is a major achievement because this order is robust against network delays, tampering and other attacks.

However, proof-of-work blockchain are not truly tamper-proof. The tamper-resistance is ensured through mining power. As long as more than 50% of the network's CPU power is controlled by honest agents, the correctness of the system is ensured. However, if any attacker is able to obtain a majority of computing power, any tampering can be possible because the attacker can mine invalid blocks faster then the honest minority. For very large networks such as the Bitcoin network, the probability of a single attacker having a majority of CPU power is very small. However, smaller networks or collusion attacks in which multiple miners work together can be successful as shown by some examples [**? ?**].

The majority of the mining power can also be used to create a intentional fork in the chain in order to



(a) Conceptual representation



(b) Transaction block, data representation

Figure 3.2: A single block in the TrustChain fabric. Incoming arrows represent block hashes from the previous blocks of the agents' chains. Source: *Creation of TU Delft BLockchain Lab*

fix software bugs. If a majority of miners update their software to version that creates and verifies blocks differently while a minority continues with the old software, a hard fork is created such that two version of the blockchain are continued. Examples of this are Bitcoin Gold [3] and Ethereum Classic [4].

3.1.4. Scalability

A single global ledger like Bitcoin seems like a valid implementation for recording transactions for a trust system. All transactions are exchanged with all agents on the network. The blockchain creates a totally ordered set of all global transactions which intrinsically means that each agent's transactions are also fully ordered. Double spend and forks are detectable through global consensus.

However, the proof-of-work mechanism leads to issues in scalability. It restricts the global throughput to at best 60 transactions per second [**?**]. This is not enough to power a global-scale distributed trust system as we envision it. The proof-of-work consensus algorithm also leads a to a large expenditure of energy which creates large transaction costs. This is not acceptable if the trust system should be for general purpose applications. For example, users will not accept to pay for each video they stream on the Tribler platform. A different solution is therefore needed.

In Section **??** several improvements to the scalability are mentioned. Examples are alternative consensus protocol such as proof-of-stake and delegated proof-of-stake. Also, sharding and off-chain transaction channels have been proposed which are supposed to increase throughput. Each solution balances three design variables: decentralization, scalability and consistency. This is commonly called the scalability trilemma in blockchain communities [5]. Crytpocurrencies generally target decentralization and consistency as the driving design factors. Financial transactions have high value and require consistency. Secure payments are enabled while scalability is less important.

3.2. TrustChain

TrustChain's design focus is in stark contrast with that of Bitcoin. Instead of employing global consensus to guarantee a single global ledger, all agents in the TrustChain fabric are owner of a chain for themselves. This creates a scalable solution in which the total network throughput grows with the size of the network. However, the scalability comes at the cost of consistency guarantees.

We argue that when designing a global-scale trust system, interaction throughput has higher value than consistency. Although undesired, we consider the manipulation of an agents reputation as less severe than stealing large amounts of money. Therefore, an architecture such as TrustChain with unbounded scalability is a better basis for a trust system than single global ledgers.

We continue this chapter with a description of the TrustChain architecture.

[3]https://bitcoingold.org
[4]https://ethereumclassic.org
[5]For example mentioned in the Ethereum wiki. https://github.com/ethereum/wiki/wiki/Sharding-FAQs

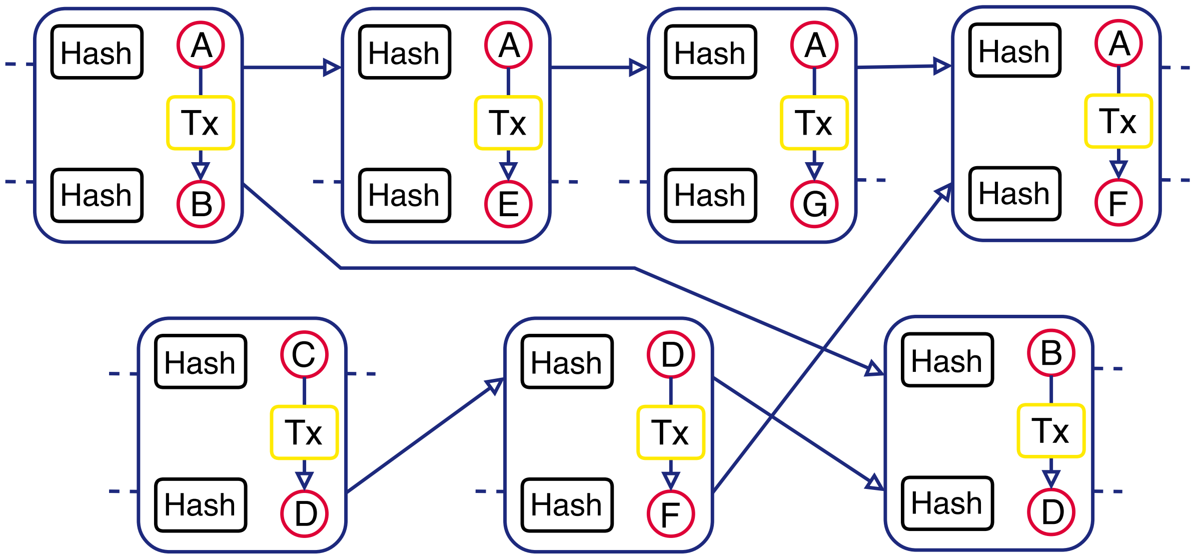


Figure 3.3: TrustChain graph structure with multiple transactions. The top row shows the chain of an agent A with links to multiple other blocks. Source: *Creation of TU Delft BLockchain Lab*

3.2.1. Data structure

Similar to the Blockchain architecture TrustChain records transactions in blocks and links blocks with the use of hash pointers. Though, in contrast to having a single chain of blocks for the whole network, in TrustChain each agent starts with an own genesis block. Hence, all agents record their own transactions on their chain.

Blocks in TrustChain contain exactly one transaction between two parties. Because both parties have their own chain and the transaction concerns both parties, any new block is added to both agents' chains. Each block contains the following main elements:

- **Transaction.** The transaction field records the value that was exchanged between agents. TrustChain is designed to be application agnostic. Thus the content of a transaction can be any serializable data.
- **Previous block hashes.** The hashes of the previous blocks of both agents' chains firmly attaches the new block to their history. This is similar to the basic blockchain concept.
- **Public keys.** In order to uniquely identify the two agents that conduct a transaction their public keys are recorded.
- **Signatures.** Both agents provide a digital signature of the transaction with their private key which any agent can check with the agents' public keys. This authenticates the transaction and cryptographically proves that the real owners of the private key conducted the transactions.
- **Sequence numbers.** All blocks on an agent's chain have a unique sequence number which shows the position in the chain.

A simplified depiction and data representation of a TrustChain block is shown in Figure **??**. When looking at a single agent the given data structure creates a chain of blocks which describes all the transactions of that agent. However the second incoming and outgoing edge of each block entangles an agent's chain with those of all partners. When looking at a complete network, of the transactions the represents a directed acyclic graph(DAG). Such a structure is shown in Figure **??**.

3.2.2. Verification protocol

TrustChain blocks are designed in a way to be verified by any other node. The verification is required to ensure that the block has not been tampered with and is correctly connected to a valid chain. A block can only be valid if all of the following conditions are true:

1. The transaction field contains the necessary information to properly define a transaction in the application context. For Tribler it needs to contain at least the amount of data which was uploaded and downloaded between the two agents.

2. Both block hashes need to point to the valid, previous block on the chain of the same public key. This makes the validity a recursive definition which depends on the previous content of the chain of both agents.

3. Both public keys need to be valid public keys.

4. The signatures need to be correct for the given public keys and block content.

5. The block needs to carry the next integer as sequence number compared to the blocks that the previous hashes point to.

The true validity of a block can only be established if the verifier also checks all other blocks on the chains of both parties. Yet, that implies to check all other blocks of their partners and their partners. Such a strong form of validity can thus only be ensured by verifying an exponentially growing amount of blocks.

A weaker form of validity can be that the previous blocks pointed at, exist. That does not ensure the validity of the complete chain but allows for a simple non-recursive validity check.

3.2.3. Tamper-evidence

Blocks of the TrustChain architecture are tamper-evident. Blocks are chained through hashes of their content and signed by both transaction parties. Any change to the content of the block, that is the transaction field, the sequence numbers, the signatures, previous hashes or public keys will change the hash of the block. Also, any such change will invalidate the original signatures of both agents. An agent can renew her own signature but not the signature of the partner without access to the partner's private key. Apart from the invalid signatures also any consecutive block's previous hash will be invalid because the tampered block's hash has changed.

We can conclude that any tampering of a block will be evident to an agent that performs verification of the signatures and hash pointers of consecutive blocks.

3.2.4. Scalability

For a transaction to be successful in TrustChain, only the involved agents need to agree on the content. This means that only two agents need to communicate to confirm a transaction. Transactions do not need to be publicly announced or sent to all nodes on the network. Also, multiple transactions can be performed at the same time globally without breaking the rules of the system. In contrast to the Bitcoin system, only the transactions of each agent are strictly ordered, the global set of transactions is not.

This results in a system that scales with the size of the network. A simple example makes this very clear. Consider two networks, one with 10 nodes and one with 100 nodes. For simplicity we consider time to be discrete and agents to interact in rounds. Each transaction takes 1 round and all agents interact. Throughput calculation is then straightforward. Nodes arrange in pairs and perform a transaction. The small network will have 5 pairs and thus 5 transactions per round, while the large network has 50 pairs and transactions. Although this example is extremely simplified, no real-world effect like network delays, network churn or bandwidth restrictions greatly conflict with this property.

TrustChain does not include a global consensus and thus does not expend bandwidth, storage and computational power to create a global order for transactions. That removes the most costly component from the general blockchain architecture. Without the expense of CPU power to calculate hashes as in the proof-of-work mechanism, transactions only cost the bandwidth and computational expense of communicating with the direct interaction partners. TrustChain enables free transactions in a global distributed system with scaling throughput based on the network size. Hence, TrustChain is valid solution for a global trust system.

3.2.5. Security

The superior scalability of the TrustChain architecture compared to single global ledgers comes at the expense of some security guarantees.

Similar to a single Blockchain the graph structure of TrustChain creates a tamper-proof record of transactions. A large difference is that each agent reigns over a chain. Each agent seemingly is able to reorder transactions but blocks are signed by a second agent and the signature will become invalid if the blocks are tampered with. Also blocks contain the previous hash of the counter party's previous block. That ensures that the hashes cannot change without the other agent being able to detect the tampering.Therefore transactions are still recorded in a tamper-proof data structure.

Still, detectabillity does not ensure detection. The can only be secured if agents continuously verify their partners data. Also, as the validity of a block is recursively defined, agents can only ensure the validity of a partners previous transaction if they possess their complete chain and verified that chain. It is even harder to defend against attacks in which agent do not directly manipulate the data structure but make use of the incomplete knowledge which is intrinsic to distributed networks.

3.3. Security analysis

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. Block manipulation

Block manipulation has been introduced previously. An agent changes a property of a block in his chain, distorting the true history in order to gain an advantage. For example, a block records a transaction in Tribler in which agent A downloaded from agent B. The transaction reduces the reputation of agent A which is why A could decide to change the amount of data downloaded to 0. If agent A does this before the signatures are provided, B will not agree to sign the transaction and will ignore A in the future because A obviously is not an honest partner. On the other hand, if both agents sign the transaction first, A can change the value on his own chain without B immediately noticing. The signature of B however becomes invalid because B signed the original, correct version of the block. In any following transaction, any agent C that is about to interact with A has the chance to detect this fraud by checking the signatures on all previous transactions of A.

TrustChain makes any manipulation detectable. Still if agent C collaborates with agent A or agent C is "lazy" and does not obtain A's chain to verify it prior to an interaction, C can perform a transaction. The collusion or laziness cannot be detected and C will remain honest in the eyes of future partners.

3.3.2. Forking and double-spending

Forking is one of the most well-known attacks of blockchain based systems. In the cryptocurrency context it is also known as double spending. An attacking agent performs two conflicting transactions with two different partners. This translates to two transaction blocks which have the same sequence number for an attacking agent A. Partner B and C are each not aware of the other version of the block and will sign the block. This allows to "overwrite" a negative transaction (with B) with a positive transaction (with C). A will only keep the positive transaction on the chain.

TrustChain makes this attack detectable through the sequence number of the blocks. If any agent obtains both versions of the block, that agent will see that A has signed and thus authorized the creation of two blocks with the same sequence number. Also if B obtains any later blocks from A the hashes will not point to the transaction with B. This creates a *proof-of-fraud*.

The actual detection of this attack requires agents to obtain transactions from their peers and compare those to existing blocks. Because no agent is aware of the transaction with B, agents cannot not specifically request B's history to check for the attack. Therefore, the detection requires the collaboration of the network in the form of random block requests. As we will show in Chapter **??**, without exchanging information about their transactions, agents are not able to find a double-spender.

3.3.3. Block withholding

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 others have in the attacker. The architecture of TrustChain makes such an attack easily detectable. The blockchain of any agent creates a tamper-proof, irreversible order for all transactions. Only if the sequence numbers increase exactly by one to each following block can another agent be sure that the whole history was shared by an agent. Therefore to establish the trustworthiness of an agent, another agent at least needs to obtain their full chain.

Note that if an agent witholds the latest blocks, this cannot be detected by another agent (except if that information was obtained already through gossiping). Yet such an attack is similar to a double-spend because any newly created block will conflict with those blocks that were withheld.

3.3.4. Whitewashing

Agents that have a significantly bad reputation such that finding interaction partners becomes difficult 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 new agents need to “pay their dues” before being accepted by the network as equals. For example, assume honest agents only upload data to agents that are above a certain level of reputation. Once an agent is below that boundary he needs to upload data before downloading again. Also, new agents start at that boundary level of reputation. In that case whitewashing will not be possible. However the question then remains how the system can be bootstrapped and maintained active because with each new agent the overall network reputation decreases such that at some point only uploading is possible but no agent is able to download.

3.3.5. 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 fake agents, called the Sybils. Together with the attacking agent the set of agents is called a Sybil region in the network. The Sybils create transaction blocks 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. Each Sybil has a valid public and private key and is able to sign transaction blocks. They create valid transaction data without any neccessary manipulation or proof-of-fraud.

Multiple ways have been explored to prevent this attack. One way is to analyze the network topology and use it to detect Sybil regions. A trust mechanism such as NetFlow, which has been proposed in [**?**] is resistant against weakly beneficial Sybil attacks. Another way is to increase the cost of registering new identities in the system. If each registered public key needs to be anchored on a costly blockchain like Bitcoin, it becomes much harder to create a Sybil region. We leave this problem to future research.

3.3.6. Collusion

Attacks are called a collusion attack if multiple attackers work together to achieve a beneficial situation for at least one of the colluders. Attacks such as those described above, can become more successful when multiple attacks collude. An example of this was given for block manipulation. If one agent manipulates a block and afterwards interacts with a colluder to share the wrongly obtained positive reputation, the colluders chain seems completely valid to honest agents.

In TrustChain it is hard to detect such collusion because we do not know whether agents actually did not know about an attacker or just claim so. We will next look at an extended system which records the knowledge of agents. Although this creates protection against some collusion attacks, still not every collusion can be protected against.

3.4. Chapter conclusion

TrustChain is a scalable solution for tamper-evident recording of transactions in a distributed network. It therefore fulfills many of the requirements that we set for our system in the previous chapter. However, TrustChain cannot in itself provide consistency of interactions as we have shown with the discussion of the double-spend attack. We show in the next chapter that gossiping is a valid solution to solve this problem. Still TrustChain provides no strong incentive for agents to gather blocks from their peers. This opens the system to free-riders who do not exchange information and verify blocks but remain honest in the eyes of their peers.

Additionally, the lack of incentive for gossiping decreases the probability of indirect reciprocity because agents are not aware of each others good or bad reputation. As we have discussed in the previous chapter, global-scale trust system will mostly rely of indirect reciprocity. Therefore the dissemination of data is essential to our system.

On the other hand, such a strong exchange policy leads to a challenging storage and bandwidth requirement. Each agent’s subjective network state will approximate the complete networks data.

4 Formal model

We now aim to show how the consistency and dissemination of encounters can be ensured. For that purpose we define a model for interactions and their dissemination in a network of agents which resembles the TrustChain system. We also define a mechanism to record information exchanges which allows to protect the network against free-riders who do not acquire and disseminate interaction records. Our proposal is to introduce *exchange transparency* which can effectively punish free-riders. We further show that it is possible to detect agents that consciously interact with malicious agents, exposing them as verification free-riders or accomplices.

4.1. Model definition

We make use of the ordered interaction model which has been introduced in previous work by Otte et al. [**?**]. We restate it in Definition **??** for clarity. The symbols were annotated in order to reuse symbols in later definitions.

Definition 1 (Ordered interaction model)**.** . An ordered interaction model $\hat{M} = \langle \hat{P}, \hat{I}, \hat{a}, \hat{w} \rangle$ consists of two sets and two functions.

- $\hat{P}$, a finite set of agents
- $\hat{I}$, a finite set of interactions
- $\hat{a} : I \to P \times P$, a function mapping each interaction to the agents involved in it.
- $\hat{w} : I \times P \to \mathbb{R}_{\geq 0}$, a function which describes the contribution of an agent in an interaction

The model allows for the analysis of any type of application in which a network of agents performs transactions that can be described by a quantitative amount. A more hands-on application of this model is Tribler. Similar to our description in Section **??**, agents model Tribler instances, interactions are data transactions on the BitTorrent network where w describes the net amount of data uploaded or downloaded. Also, interactions can in practice be recorded by TrustChain blocks which store the involved agents $a(i)$ and the amount of data $w(i)$.

However, the model does not explicitly model the information exchange between agents. Our goal is to show that in a distributed trust system this exchange of information is also an essential component to guarantee manipulation resistance. We therefore extend the model with exchanges.

Definition 2 (Ordered encounter model)**.** An ordered encounter model $M = \langle P, I, E, a, w, x \rangle$ is a 6-tuple consisting of three sets and three functions. For convenience we define a set $N = I \cup E$ as the set of all encounters.

- P, a finite set of agents
- I, a finite set of interactions
- E, a finite set of exchanges

- $a : N \to P \times P$, a function mapping each interaction and exchange to the agents involved in it.
- $w : I \times P \to \mathbb{R}_{\geq 0}$, a function which describes the contribution of an agent in an interaction
- $x : E \times P \to \mathscr{P}(N)$, a function which describes the set of encounters that an agent learns about in an exchange. $\mathscr{P}(N)$ denotes the power set of N.

Note that for any interaction $n \in I$ in which $p \notin a(n)$, $w(i, p) = 0$ must hold and similarly for any exchange $e \in E$, $x(e, p) = \emptyset$ if $p \notin a(e)$.

An encounter $n \in N$ happens between two agents for one of two reasons: to perform a transaction of some value or to exchange information. The function x describes a set of encounters that an agent obtains knowledge about through an exchange. A gossip protocol, that is the exchange of information with a random peer, is one way to create such exchanges. We will formally introduce the knowledge concept in Definition **??**.

For each agent we can describe the set of encounters that agent was involved in as the agent encounter history.

Definition 3 (Agent encounter history)**.** Given an ordered encounter model $M = \langle P, I, E, a, w, x\rangle$ with encounters N the agent encounter history of an agent $p \in P$ is defined as follows:

$$H_p = \{n \in N : p \in a(n)\} \tag{4.1}$$

In a similar way the set of the agents interactions I_p and E_p can be defined.

$$I_p = \{n \in I : p \in a(n)\} \tag{4.2}$$

$$E_p = \{n \in E : p \in a(n)\} \tag{4.3}$$

The history of any *honest* agent p is totally ordered by $<$. If we denote the i-th encounter of p as $n_p(i)$ then we can alternatively write the agents history of length t as $H_p(t) = \{n_p(1), n_p(2), ..., n_p(t)\}$. Similarly we can define $I_p(t)$ and $E_p(t)$ as the interactions and exchanges of p after t encounters. Note that $N_p(t) = I_p(t) \cup E_p(t)$. A *dishonest* agent can break the order of their history. We will define this behavior in Definition **??**.

In our model encounters between agents are not public. This creates a discrepancy between the set of all encounters N and the known encounters from an agent p's point of view. Agents only have knowledge of those encounters which they were involved in or which they obtained knowledge of through an exchange. We can then define the subjective network state, which is the knowledge an agent has about the state of the network.

Definition 4 (Subjective network state)**.** Given an ordered encounter model $M = \langle P, I, E, a, w, x\rangle$ and an agent $p \in P$ with encounter history H_p, the subjective network state of agent p is defined as follows:

$$N_p = H_p \cup \{x(n, p) : n \in E_p\} \tag{4.4}$$

Again we can denote the subjective network state of p after t encounters as $N_p(t)$. The subjective network state contains the complete knowledge of the agent. Otte et al. define a similar concept in the form of the subjective work graph. It generally models a partial view of the network described by the model. This subjective view is the basis for their calculation of reputation and trust. The relation to the creation of trust is further discussed in Section **??**.

Agents can only reason about their subjective network state. Therefore an agent p can only *evaluate* the function $x(e, q)$ if $e \in N_p$ and $x(e, q) \subset N_p$.

The model from Definition **??** introduces exchanges. Yet it does not define how agents should exchange data. We therefore define an exchange policy. Any honest agent exchanges encounter information according to a given exchange policy.

Definition 5 (Exchange policy)**.** An exchange policy is a function $f : P \times P \times \mathbb{Z} \times \mathbb{Z} \to \mathscr{P}(N) \times \mathscr{P}(N)$. For two agents p and q with histories $H_p(i)$ and $H_q(j)$ who would like to interact, an exchange policy f defines the sets $N^p \subseteq N_p(i)$ and $N^q \subseteq N_q(j)$ that have to be sent by p and q, respectively. We write $f(p, q, i, j) = (N^p, N^q)$.

Our goal is to ensure that any honest agent adheres to the exchange policy defined for the network. This will allow us to ensure that interactions are well disseminated and that agents cannot free-ride on storing information.

As the exchanges are part of an agents history it is possible to determine whether an agent has always exchanged in accordance with a certain exchange policy.

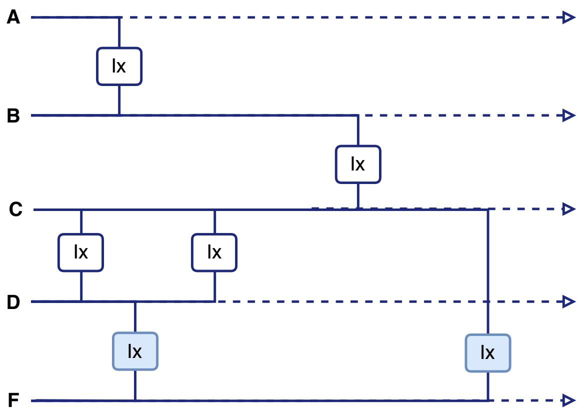


Figure 4.1: History graph of a network No-Exchange policy in our model. We consider 5 agents. Each agent has a time line which is connected to interactions(denoted by Ix). Each interaction is between two agents. The blue blocks show the subjective network state of agent F.

Definition 6 (Exchange policy adherence)**.** Given an agent p, p adheres to the policy f if for every interaction $n(k) \in I_p$ with partner agent q there exists a unique exchange $n(i) \in E_p(k-1)$ such that $N^q \subseteq x(p, n(i))$ and $N^p \subseteq x(q, n(i))$, where N^q and N^p are defined through $f(p, q, i-1, j-1) = (N^q, N^p)$, j being the length of q's history before the exchange.

4.2. Fork and double spend defense

In what follows we analyze the defense of certain exchange policies against a double spend attack. That attack is a potent attack on a network that can be modelled by an ordered encounter model. Conceptually, an attacker has two conflicting interactions at the same time with two different agents. Each partner is not aware of the other, therefore both accept the interaction.

Definition 7 (Fork and double spend)**.** Given an ordered encounter model $M = \langle P, I, E, a, w, x\rangle$, a malicious agent p has two conflicting interactions $i \in I$ and $j \in I$ at the exact same time such that $i = n_p(t)$ and $j = n_p(t)'$. Both interactions have a different partner $a(i) = (p, q)$ and $a(j) = (p, r)$. Accordingly, the history of p is not a totally ordered set anymore. The attack can be detected if for any honest agent s, $i \in N_s$ and $j \in N_s$ are true.

In the following we show that exchanging blocks is essential for the network to defend against such an attack. We first study the case of no exchanges. For that we formally define the No-Exchange policy.

Policy 1 (No-Exchange)**.** *The No-Exchange policy f^{NE} defines no required exchanges for an honest agent, so for any two agents p and q and any lengths of their encounter histories i and j, $f^{NE}(p, q, i, j) := (\emptyset, \emptyset)$.*

Without any exchanges, agent are only aware of the interactions that they perform themselves. In Figure **??** we visualize a situation in which five honest agents perform interactions. The timeline of each agent is represented by the horizontally dashed line. Each interaction is connected to two agents' timelines. The No-Exchange policy requires no information exchange so agents only have interactions. Each connected interaction is part of an agent's history. The figure indicates with blue, the blocks that agent F is aware of which are only those he was involved in.

Theorem 1 (Detectability of double spend without exchanges)**.** *If honest agents apply the No-Exchange policy, the fork and double spend attack cannot be detected by any honest agent.*

Proof. Let p be the attacking agent and let i and j be the conflicting interactions of the attacks. Further, let q be the partner in interactions i such that $a(i) = (p, q)$ and let r be the partner of interaction j such that $a(j) = (p, r)$. The No-Exchange policy implies that for any honest agent $s \in P$ and all $e \in E$, $x(e, s) = \emptyset$ and therefore $N_s = H_s$ applies. According to Definition **??** $i \in N_q, j \notin N_q$ and $i \notin N_r, j \in N_r$. Furthermore, for any honest $s \in P$, $i \notin N_s$ and $j \notin N_s$ must be true because $s \notin a(i)$ and $s \notin a(j)$. □

Theorem **??** proves that the double spend attack cannot be detected without exchanging encounters.

We now show that if agents do exchange information, this can lead to the detection of the attacker. We define an exchange policy in which honest agents obtain the encounter history prior to an interaction.

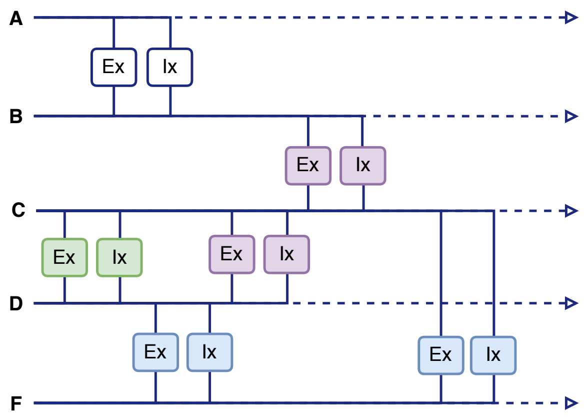


Figure 4.2: History graph of History-Exchange policy in our model. We again consider 5 agents. The colored blocks show the subjective network state of agent F. The blue blocks are F's own encounters, the green block is an encounter received from D while the purple blocks are received from C.

Policy 2 (History-Exchange policy)**.** *The History-Exchange policy f^{HE} requires for any two agents p and q and any lengths of their encounter histories i and j:*

$$f^{NE}(p, q, i, j) := (H_p(i), H_q(j)) \tag{4.5}$$

Figure **??** shows the same 5 agents, with the same interactions. This time agents apply the History-Exchange policy, which requires the exchange of both agents' histories prior to an interaction. The colored interactions and exchanges are received by F, green being those received from D and purple those received from C. At the end of the shown timeline, the colored blocks are the subjective network state of F.

Theorem 2 (Detectability of double spend with the History-Exchange policy)**.** *If all honest agents apply the History-Exchange policy and periodically use uniform sampling over the set of agents to find an interaction partner, the double spending will eventually be detected.*

Proof. Let p be a malicious agent and let $i \in I$ and $j \in I$ be the conflicting interactions of the attack. Further, let q be the partner in interaction i such that $a(i) = (p, q)$ and let r be the partner of interaction j such that $a(j) = (p, r)$. According to Definition **??** $i \in N_q, j \notin N_q$ and $i \notin N_r, j \in N_r$.

We consider an honest agent q sampling agents for an interaction after the double spend. For network size $l = |P|$ the probability of choosing partner r in the first round is $1/l$. The probability for not having chosen r in any of k rounds is $(\frac{l-1}{l})^k$ and $\lim_{k\to\infty}(\frac{l-1}{l})^k = 0$. Given an infinite amount of time, q therefore chooses r for an interaction k and because q is honest they do an exchange e prior to their interaction.

Before the exchange, $j \in H_r$ holds. Both r and q are honest and adhere to the exchange policy f^{HE}, therefore $j \in x(e, q)$ holds. This means that after the exchange e, $i \in N_q$ and $j \in N_q$ must hold. Therefore q is able to detect the double spending. □

Theorem **??** shows that exchanging encounter information makes double spending detectable. Once a double spender is detected by an agent it is possible to ignore them for future interactions. The amount of exchange rounds until the detection depends on the exact policy of exchanging data and a deeper analysis is beyond the scope of this work. However, qualitatively, exchanging more information in each exchange obviously leads to faster detection.

4.3. Exchange free-riding

We now look at ways to incentivize information exchanges. At this point we have shown that in a network in which interactions are only directly observable by the involved agents, an exchange mechanism like the History-Exchange policy is essential to defend against the double-spending attack. Our aim is now to show that such an exchange policy can be ensured. Conceptually, if the exchange behavior is visible in the history of an agent, any missing or incomplete exchange will uncover that agent as a fraud.

We consider an incentive not to exchange encounter information. If bandwidth and storage capacity are valuable resources for agents, data dissemination and their long-term storage come at a cost for agents.

Therefore agents can attempt to be lazy and only interact without exchanging the necessary data. However if all agents would behave in that way the system loses the ability to defend itself against forks and double spending as shown in Theorem **??**. We will call those lazy agents *exchange free-riders*.

Definition 8 (Exchange free-rider)**.** Given an exchange policy f that all honest agents use, an exchange free-rider is a dishonest agent p whose encounter history H_p does not adhere to policy f as defined by Definition **??**.

Any failure to send or receive the correct encounter information before an interaction will make an agent an exchange free-rider. We now show that any honest agents will not commit to an interaction with an exchange free-rider

Theorem 3 (No interaction without correct exchange)**.** *An honest agent who adheres to the History-Exchange policy can not interact without a successful exchange except with a double-spender.*

Proof. Given the History-Exchange policy f^{HE}, an honest agent p with encounter history $H_p(i)$ and a potential partner q with history $H_q(j)$, p will request an exchange e, such that $e = n_p(i+1) = n_q(j+1)$. f^{HE} further defines that p needs to send $H_p(i)$ and q needs to send $H_q(j)$. Also, because p is honest, p will send the correct data such that $H_p(i) \subseteq x(q,e)$.

The correctness of the exchange depends on the behavior of q. Only if q also sends the correct information such that $H_q(j) \subseteq x(p,e)$, then p and q do comply with the exchange policy f^{HE}. If p finds that $H_q(j) \subseteq x(p,e)$, p can conclude that either q has sent the correct encounters, or q is attempting a fork if there exists $n_q(j+1) \notin x(q,e)$. The latter cannot be prevented.

p can verify whether $H_q(j) \not\subset x(p,e)$ if there exists some $k \in \mathbb{Z}$, $1 \leq k \leq j, n_q(k) \notin x(p,e)$. In order to not create proof of being an exchange free-rider, p cannot interact with q if p finds the exchange to be incomplete. □

According to Theorem **??** any honest agent who applies the history exchange policy can ensure that no free-riding happens in their encounters except if their partner attempts to fork. Note that the architecture we present cannot prevent the creation of forks, however eventual fork detection has been proven in Theorem **??**. Only with collusion two agents can attempt to not exchange the correct data and still interact. Such historical free-riding with third-parties cannot be detected with the History-Exchange policy. In the next section we define a policy that does allow this.

The incentive to not commit an interaction without a complete exchange is created through the recording of the exchange on the history of the agent. If an agents history proves an incomplete exchange before an interaction, the agent does not adhere to the exchange policy anymore according to Definition **??**. We show in the next section that if any agent has the complete subjective network state of another agent, the complete history of an agent can be verified to no include any free-riding on the exchanges.

4.4. Verification free-riding

We have shown that forks will eventually be detected and that free-riding is not possible in an encounter with an honest agent. However, free-riding in other encounters cannot be detected and agents who do not perform or ignore consistency checks, and thus interact with double spender cannot be detected.

In order to solve those two problems, a stronger exchange policy can be enforced. This will allow to audit the complete history of another agent and detect any inconsistencies or incorrect behavior.

In order to perform such a deep analysis, honest agents need to obtain the full subjective network view of another agent before each interaction. Therefore we define the Network-State-Exchange policy.

Policy 3 (Network-State-Exchange)**.** *The Network-State-Exchange policy f^{NSE} requires for any two agents p and q and any lengths of their encounter histories i and j:*

$$f^{NSE}(p,q,i,j) = (N_p(i), N_q(j)) \tag{4.6}$$

In order to visualize the difference to the other exchange policies, we show the same 5 agents in Figure **??**, with the same interactions. This time agents apply the Network-State-Exchange policy, which requires the exchange of both agents' subjective network state prior to an interaction. In contrast to the History-Exchange policy, this time F also received knowledge about interactions from agents A and B. The policy greatly increases the storage requirements. We will consider this in Section **??**.

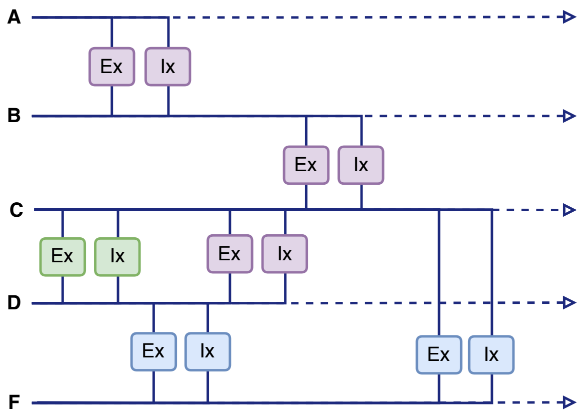


Figure 4.3: History graph of Network-State-Exchange policy in our model. We again consider 5 agents. The colored blocks show the subjective network state of agent F. The blue blocks are F's own encounters, the green block is an encounter received from D while the purple blocks are received from C. This time F obtains the complete network state.

We first show that if an agent p knows the subjective network state, that is the full knowledge of another agent q, p is able to determine whether q adhered to the Network-State-Exchange policy.

Lemma 1 (Verification of adherence to policy)**.** *Given the subjective network state $N_q(t)$ of an agent q, an agent p is able to determine whether that agent adhered to the Network-State-Exchange policy.*

Proof. For every interaction $n_q(i) \in I_q(t)$ with a partner $s \in P$ there should exist a unique exchange e such that $a(e) = (q, s)$ and $e = n_q(j) = n_s(k)$. If the exchange is correct $N_s(k-1) \subseteq x(e, q)$ and thus $N_s(k-1) \subset N_q(t)$.

If p knows $N_q(t)$, p can evaluate whether $N_s(k-1)$ is in $N_q(t)$ because $H_s(k-1)$ must also be in $N_q(t)$. According to Definition **??**, $N_s(k-1)$ can be inferred from the history $H_s(k-1)$ by evaluating $x(n_s, s)$ for all exchanges $n_s \in E_s(k-1)$. If for any transaction partner s of q, p cannot evaluate any $x(n_s, s)$, that means that some $n \in x(n_s, s)$ is not in $N_q(t)$. In that case q cannot be honest because q can also not evaluate $x(n_s, s)$. □

The previously defined lemma allows us to prove that any honest agent, in a network that applies the Network-State-Exchange policy, can verify whether their partner has up to the point of their encounter adhered to the policy.

Theorem 4 (Detectability of free-riding)**.** *An honest agent who applies the Network-State-Exchange policy is able to verify whether another agent is a free-rider.*

Proof. Let p be an honest agent who applies the Network-State-Exchange policy and let q be an agent that exchanged encounters with p. Let e be their latest exchange such that $e = n_p(i) = n_q(j)$ then $N_q(j-1) \subset N_p(i)$ should hold.

p can ensure that $x(e, p) = N_q(j-1)$ because $N_q(j-1)$ includes $H_q(j-1)$ and $N_q(j-1)$ can be inferred from H_q by evaluating $x(n_q(k), q)$ for all exchanges $n_q(k) \in E_q(j-1)$, $k < j-1$. If any $x(n_q, q)$ cannot be evaluated by p, this means that $N_q(j-1)$ is not complete. Otherwise it holds that $N_q(j-1) \subset N_p(i)$.

According to Lemma **??** p is then able to verify whether q adhered to the Network-State-Exchange policy. □

This creates a stronger guarantee than Theorem **??** as even historic dishonest behavior is detected and leads to that agent being ignored.

The Network-State-Exchange behavior further allows any agent p that exchanged information with an agent q to reconstruct the subjective network state of agent q after any encounter j. This makes it possible for an honest agent p to re-evaluate q's decisions to engage in each of q's interactions.

Theorem 5 (Detectability of interactions with double-spender)**.** *An honest agent who applies the Network-State-Exchange policy is able to detect any wrongful interaction with a double spender.*

Proof. Let p be an honest agent who applies the Network-State-Exchange policy and let q be an agent that exchanged encounters with p. Let e be their latest exchange such that $e = n_p(i) = n_q(j)$ and because p is honest, $x(e, p) = N_q(j-1)$ and $x(e, q) = N_p(j-1)$. Therefore it holds that $N_q(j-1) \subset N_p(i)$. This implies also that any previous subjective network state of q, so for any $1 \leq k \leq j-1$, $N_q(k) \subset N_p(i)$. Then, for all k, p can determine the set of double-spender (malicious) $P^m(k)$ agents that q was aware of after k encounters. If there exists an interaction $n_q(l)$ with some malicious agent $s \in P^m(k)$ and $l > k$, p finds that q knowingly interacted with a malicious agent or did not perform the verification to find s. Both cases are considered dishonest behavior. □

We have shown in Theorem **??** that the Network-State-Exchange policy enables agents to effectively find any wrongful interactions with double spenders. If p finds q to be wrongful, q either colluded with a double spender or q did not find the double spender which means q did not perform the right verifications. In the latter case we call q a verification free-rider.

We can combine our findings in an audit algorithm which allows an honest agent p to establish the correctness of another agent q's encounter history.

Algorithm 1 Audit algorithm which verifies that an agent has adhered to the Network-State-Exchange policy and has not knowingly interacted with a malicious agent

```
procedure HISTORYAUDIT($N_q(t)$)
    $P^v \leftarrow \emptyset$, set of valid next interaction partners
    $P_m \leftarrow \emptyset$, set of detected malicious agents
    $N_q(0) \leftarrow \emptyset$, $q$'s subjective network state
    for all encounters $n(i)$ in $N_q(t)$ do
        $s \leftarrow$ partner in $n(i)$ such that $a(n(i)) = (q, s)$
        $N_q(i) \leftarrow N_q \cup \{n(i)\}$
        if $n(i)$ is an interaction then
            if $s$ is in $P^m$ then
                return false
            else if $s$ is not in $P^v$ then
                return false
            else
                $P^v \leftarrow P^v \setminus \{s\}$
        if $n(i)$ is an exchange then
            $N_q(i) \leftarrow N_q(i) \cup x(n(i), q)$
            if $x(n(i), q) = N_s(j)$ then
                $P^v \leftarrow P^v \cup \{s\}$
            if $\exists r \in P, n_r(t), n_r(t)' \in N_q(i)$ then
                $P^m \leftarrow P^m \cup \{r\}$
    return true
```

At this point the question arises who verifies that agents perform this detection of verification free-riders. The detection of verification free-riders is a recursive problem and therefore will not lead to an infinite verification process. However, because agent exchange their complete subjective network state, they also share their knowledge of malicious agents and free-riders. Suppose an agent p is aware of a set of malicious agents S_p. After exchanging all information with another agent q, q's set of malicious agents should at least contain $S_p \subseteq S_q$. Should an agent q interact with any agent in S_p afterwards, at least agent p knows that q is dishonest. This way q will slowly be isolated.

4.5. Relevance for global trust system

The calculation of reputation and trust has not been part of the discussion up to now. This brings about the question of relevance to the global trust system. We refer again to the work of Otte et al. for an in depth study of interaction graph based trust mechanisms. The authors show that scoring mechanisms based on a graph representation the interactions between agents can be designed to be resilient against Sybil-attacks. For their analysis the authors use the model stated in Definition **??** as the basis for creating a graph.

The mechanisms described in this chapter improve trust systems by ensuring that the encounters which are the basis of the calculations of trust and reputation are honest, well disseminated and consistent. We mention in this section a few conclusions that we can draw from the above theoretical analysis.

4.5.1. Manipulation resistance

Instead of exploring new ways of calculating trust, we analyzed in the above the resilience to of the model in the presence of malicious and free riding agents. We have shown that in order to defend against forks and double spending, information exchange between agents is critical. Only through ensuring a proper defense against attacks can we warrant the validity of the interactions. Therefore we create an exchange mechanism that aligns the exchange behavior with an incentive of agents. The recording of encounters leads to exchange transparency which exposes free-riders. This effectively leads to better manipulation resistance and ultimately increases the validity of reputation and trust.

4.5.2. Increased accuracy of reputation estimate

Apart from the manipulation resistance, encouraging agents to acquire more data also leads to a better estimation of the true network state. This in turn leads to more accurate calculation of the reputation of peers. According to the model of Mui [**?**] which was introduced Chapter **??** this leads to self-reinforcing cycle of more trust and increased reciprocity. An analysis of the influence of gossiping on the trust calculation with a trust mechanism such as the one introduced in [**?**] is an interesting topic for further research.

4.5.3. Pairwise agreement on reputation

The Network-State-Exchange policy leads to a synchronization of both agents subjective network state. After the exchange both agents share the same subjective network state which means they also agree on the reputation of all peers as the reputation is based on the subjective view of the network. If trust is calculated with a function that takes as input only the knowledge of reputations or the records of interactions, both agents can check their trust for each other. This way no dispute can exists about the fairness of a trust-based contribution. For example if one agent p is requested by two other agents to provide a resource, p cannot falsely donate the resource to the less trusted agent in a collusion. The agent who ended up empty handed could prove that the trust calculation from p's perspective points in his favor.

4.5.4. Reduced scalability

Enforcing an exchange policy for honest agents also bears risks. An agent is only able to interact after exchanging some information, in the most stringent case all their knowledge, with another agent. This binds a requirement for storage space and bandwidth to the ability of interacting which might be too stringent for some application contexts.

The Network-State-Exchange policy requires agents to share all their knowledge. After an exchange both agents therefore have the combined knowledge of both agents. This concept is also studied in the context of replicated databases in work by Demers et al. [**?**]. They define the anti-entropy mechanism in which a database instance randomly picks other instances to resolve any differences. Their results show that any update will eventually be distributed to all instances. Complete dissemination is reached with a time complexity proportional to the logarithm of the population size. If all agents perform periodical random peer sampling for interactions, the results should be similar to those of Demers. This means that the Network-State-Exchange requires the storage of the complete networks data, that is, all interactions and encounters.

4.6. Chapter conclusion

We have shown that exchanging information of interactions and encounters allows to detect a fork and double-spend attack. Furthermore, if we record the exchanges in a similar way as interactions and make them part of an ordered history, honest agents can ensure that their partners apply a certain exchange policy. Any agent that does not stick to the policy can be detected and dealt with accordingly.

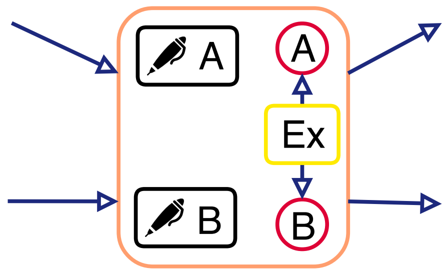
Finally, if agents apply a strong policy like the Network-State-Exchange policy, agents can re-evaluate verifications that their peers have done in the past. Any failed verification that sitll leads to an interaction will mark agents as lazy verification free-riders or accomplices.

5
Manipulation-resistant block exchange

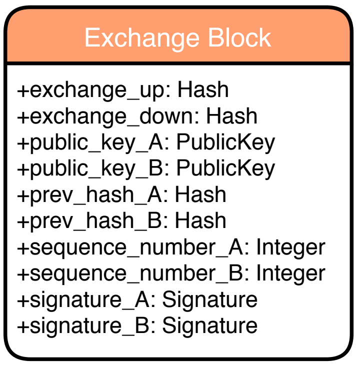
Neither a conventional blockchain solution with global consensus nor TrustChain are able fullfil all requirements of a reporting system for encounters of a trust system. Although the scalability problem has been tackled by many researchers and developers[**? ?**] no working solution has yet been proven in practice for blockchains with global consensus. On the other hand TrustChain offers a scalable solution but no strong incentive mechanism is put in place to ensure consistency and dissemination.

In the previous chapter we have shown that the exchange of encounters between agents is a defense against inconsistent data such as forks. Also, by adding reports of exchanges to the history of agents, any failure to exchange will be detectable. The recording of exchanges also makes an agent's view of the network at any given moment transparent. This can be utilized to audit any behavior in history of an agent.

In the TrustChain architecture encounters are recorded in the form of blocks. Therefore we need to enable the exchange of blocks. For the implementation of block exchanges two things are required: an *architecture* which enables the recording of exchanges and a *policy* that defines which blocks to exchange and when (defined as exchange policy in our model). We first propose an architecture based on TrustChain which allows the implementation of the complete ordered encounter model from the previous chapter. The TrustChain architecture records an agents history on a tamper-evident chain, similar to the history of the agent in our model. We describe how the architecture can be extended in order to create manipulation-resistant evidence of block exchanges. Then we describe how the Network-State-Exchange policy can be implemented.



(a) Conceptual representation



+exchange_up: Hash
+exchange_down: Hash
+public_key_A: PublicKey
+public_key_B: PublicKey
+prev_hash_A: Hash
+prev_hash_B: Hash
+sequence_number_A: Integer
+sequence_number_B: Integer
+signature_A: Signature
+signature_B: Signature

(b) Exchange block, data

Figure 5.1: A single exchange block for the TrustChain fabric. Source: *Adpated from TU Delft BLockchain Lab*

5.1. Implementation

Exchanges can be documented in a similar way as transaction data is recorded. We extend TrustChain with *exchange blocks*. Instead of the transaction field, an exchange block contains two exchange fields which store the blocks sent and received from the perspective of the agent that initiates the exchange. Similar to our model, an exchange consists of any type of block, so both transaction and exchange blocks are exchanged.

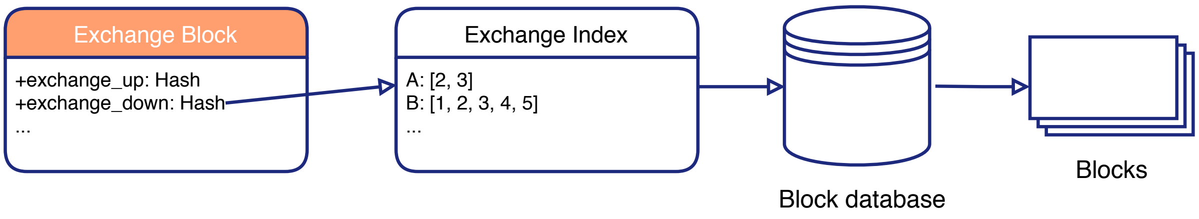


Figure 5.2: Storage of exchange information: the exchange block contains a hash, which can be mapped to an index. That index can be used to retrieve the blocks from the block database.

Exchanges can become very large, in a theoretical worst case the whole network's data. Therefore we do not store all exchanged blocks directly but only the root of a hash tree of the blocks. A conceptual representation and data representation is shown in Figure **??**. Note that except for the `exchange_down` and `exchange_up` fields the block is similar to a transaction block as shown in Figure **??**. The creation of the block is shown in Figure **??**.

Just like transaction blocks, the exchange blocks become part of an agent's chain with two incoming and two outgoing pointers. Any exchange is recorded on-chain, creating a tamper-evident history of the exchange behavior provided by the TrustChain architecture. Thus the same validity conditions apply.

Each agent keeps track of the actual set of blocks that were received in an exchange block such that the root hash of the exchanged blocks can be recomputed to check the validity of the exchange block. Specifically, each node stores an *index of the blocks* contained in an exchange in a separate database. The index maps an exchange block to the public key and sequence number of each block received. This index can be used to retrieve the actual blocks from the database of blocks. This is illustrated in Figure **??**.

With respect to our model, exchange blocks contain the agents that are involved in the exchange (represented by $a(e)$ in our model), whereas the root hashes of the exchanged blocks correspond to the function x in the model. Similar to x, the exchange hashes can only be evaluated if the block itself, and all exchanged blocks are available to an agent.

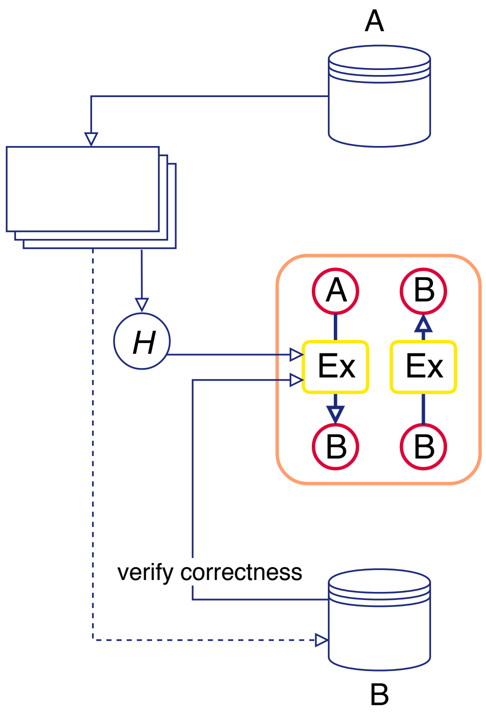


Figure 5.3: One half of a block exchange is shown. Agent A sends blocks to B. H creates the root of a hash tree of the blocks sent. The root hash is stored in the exchange block. B receives the blocks and can recompute the hash to verify the correctness. Also any other agent that has the same blocks can recompute the hash.

5.1.1. Verification protocol

The architecture is designed such that another node can validate the correctness of an exchange block. Similar to the verification of transaction blocks described in Section **??** an exchange block sequence numbers, public keys, previous hashes and signatures need to be correct.

Additionally, the hashes stored in the `exchange_down` and `exchange_up` need to be valid. An agent *A* is able to validate an exchange between *B* and *C* by requesting from *B* the blocks that *B* received and from *C* the blocks that *C* received. If *A* calculates the hash of *B*'s blocks and it adds up to the hash stored in the `exchange_down` and the same is true for *C*'s blocks and the `exchange_up` field, the block is valid. At this point *A* has also established that *B* and *C* actually stored the blocks. When *A* is only interested in confirming the honesty of one agent, for example *B*, *A* can verify only the hash of the `exchange_down`.

In order to validate all exchanges of one agent, the validator needs to acquire all blocks ever received by another agent. We showed this in Lemma **??** in Chapter **??**.

5.1.2. Storage commitment

An agent is committed to store the data that he received. As we have shown in this section any agent *A* can verify another agent *B*'s exchanges by *requesting blocks*. The verification can only be successful if *B* can summon the correct blocks. Exchanges are stored on the append-only chain where they cannot be removed. They can thus be verified also long after the exchange has taken place. In order to pass verification at any time, *B* will need to have a long-term storage for all blocks received in exchanges.

We define an exchange policy which requires agents to request blocks from each other. This ensures that block requests happen frequently and agents cannot remove any data from their database.

5.1.3. Tamper-evidence and authenticity of blocks

The tamper-evident property is inherited from the TrustChain architecture discussed in Section **??**. If a block is tampered with the original signatures of the block will be invalidated. Also any possible consecutive block's hash pointer will also be invalidated.

5.2. Implementing an exchange policy

In TrustChain the exchange behavior of agents was not visible. With the addition of exchange blocks agents can base their decisions, for example whom to interact with, on how much data peers exchanged in the past. But making the gossiping behavior visible can only be the first step. The next step is naturally to apply the exchange records to work towards a safer distributed system.

In our model from Chapter **??** we defined exchange policies, which ensure a certain exchange of information between agents. We now consider how the Network-State-Exchange can be implemented using the architecture defined previously.

5.2.1. Conceptual

According to our definition an exchange policy defines a set of blocks that are required to be exchanged prior to an interaction. Specifically, the Network-State-Exchange policy requires two agents to exchange all blocks prior to an interactions.

This strategy combines many convenient guarantees.

1. *Local consensus* is reached, such that both agents agree on a view of the network.
2. Dissemination of all transaction blocks leads to better reputation estimates.
3. Exchanges lead to detection of consistency issues.
4. No agent can free-ride in storing and disseminating data because exchange behavior is transparent on the chain.
5. Collusion becomes more difficult because also any invalid blocks will be recorded in exchanges creating *proof-of-knowledge*.
6. Audits of all behavior is possible. This enables *guilt by association*.

Each agent becomes a witness of their interaction partners. Through signing the exchange of all information they commit to storing and disseminating that information to future interaction partners. By signing a transaction after the exchange agents commit to having verified the information of their partner. Any fraudulent information that is ignored is recorded in the exchange block and can be verified in an audit as described in Chapter **??**.

5.2.2. Completeness of exchange

An obvious challenge of our architecture is how to ensure that agents actually act according to the exchange policy. We claim that any honest agent is able to verify that their partner exchanged all information.

An honest agent A exchanges blocks with B. B sends his blocks to A which includes his chain and a block index for each exchange block on the chain. A adds the received blocks to her database. If A is now able to recalculate all exchange hashes of the exchange blocks on B's chain, using the block indexes and her own database, A can be sure to have all blocks that B has. This is similar to the evaluation of function x in our model.

Note that B can possible withhold blocks from the end of the chain. A can only detect this if A has received those newer blocks in some previous exchange from a third agent C. If B withholds blocks, B is attempting to create a fork the exchange block will share a sequence number with a block that B withheld.

5.2.3. Auditing

We have shown in Chapter **??** that the Network-State-Exchange policy allows any honest agent to re-evaluate any past verification of their partner. We will now describe this process, which we call auditing. This process is the implementation of Algorithm **??**.

An honest agent A has exchanged blocks with B and verified the completeness of their exchange. A therefore has all blocks that B has. A can then rebuild B subjective network state, that is the block database of the agent, at each exchange and interaction. Going through the chain of B, for each exchange block with some agent C, A adds the blocks to the temporal subjective network state of B and checks whether B was able to recalculate the exchange hashes of C's exchange blocks. Also A checks whether all of C's blocks are valid. For every transaction block with some agent D, A checks whether there was a valid exchange with D prior to that transaction. Also A checks whether B was aware of any double spenders and still interacted with them afterwards. If that is the case B is guilty by association because he is either a colluder or did not perform verification.

If all verifications pass, A has established that B has throughout his complete history acted honestly.

5.2.4. Unresponsive agents

Each honest agent will require an exchange with their partner prior to an interaction. If an agent asks a partner to send the blocks for an exchange it is possible that the partner does not respond. This can either mean that the partner is not in the possession of the correct blocks to pass the verification and does not respond to not be exposed. On the other hand an agent can also be temporarily disconnected from the internet and therefore not able to respond. This ambiguity of not responding is not problematic for our solution. Any failure to respond will lead to no further action and therefore no transaction. If however an honest agent will at some point respond, the verification process is continued as normal.

The situation is different if an agent stops responding in the middle of an exchange or interaction. For example, A and B have exchanged data and A has sent an exchange block to B. While waiting for the signature of B, A is not able to extend the chain because once B eventually replies, any extension in the meantime on A's chain will create a fork. A practical solution is to create a timeout after which the block will be discarded. If B evetually respondes, the exchange process needs to be repeated.

5.2.5. Efficient exchange

Up to now we have considered that agents send all their blocks to each other. A simple efficiency improvement can be put in place such that only those blocks are sent which the partner does not have.

Given an agent's chain and the indexes that record which blocks were received in each exchange, any other agent can create a complete index for the database of the other agent. That index is calculated by combining the indexes of all exchange blocks and adding the transactions on the chain. The same can be done for the agent himself. The difference between the two indexes are the blocks that one agent has, but the other does not. During an exchange an agent can then specifically request only those blocks. Therefore exchanges become a synchronization in which differences between the states are resolved. Note that if both agents have different blocks of a double spend such, that block will not be synchronized with this method. However, the difference will be detected when controlling the hash.

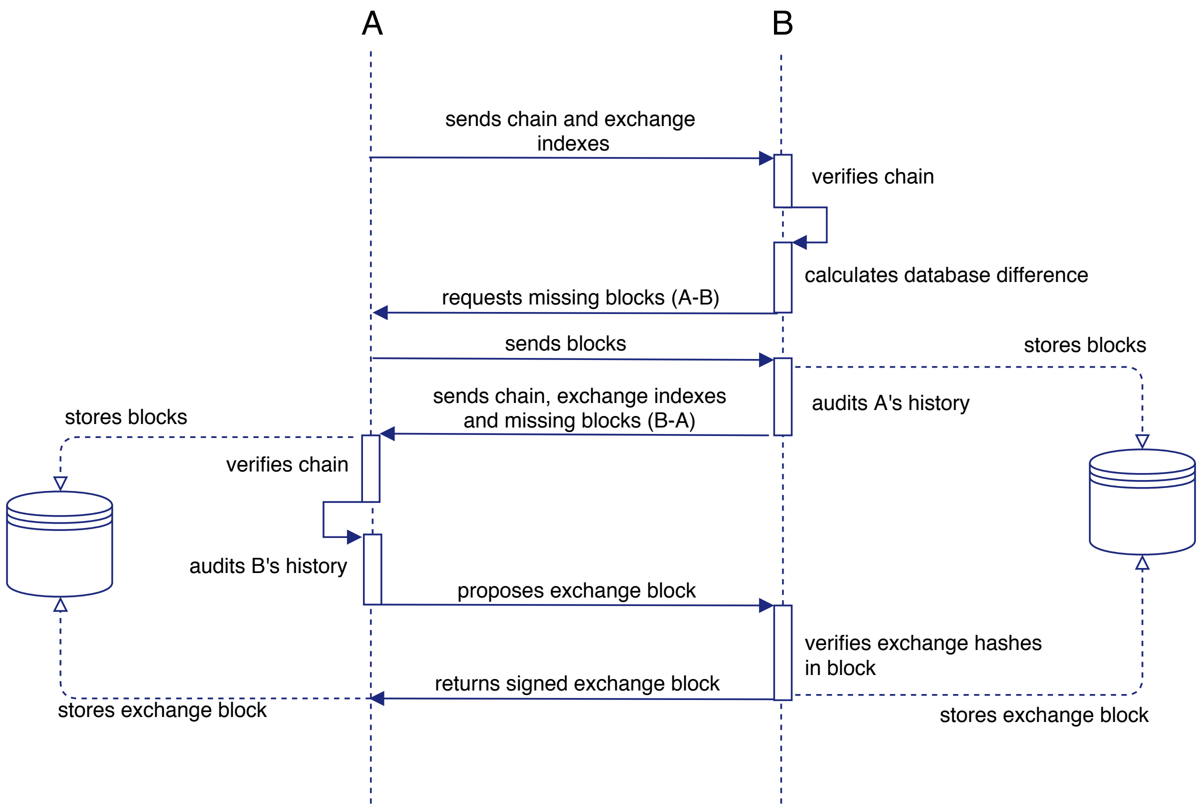

A
B
sends chain and exchange indexes
verifies chain
calculates database difference
requests missing blocks (A-B)
sends blocks
stores blocks
sends chain, exchange indexes and missing blocks (B-A)
audits A's history
stores blocks
verifies chain
audits B's history
proposes exchange block
verifies exchange hashes in block
returns signed exchange block
stores exchange block
stores exchange block

Figure 5.4: Exchange process between two agents *A* and *B*.

5.2.6. Exchange process

We shall now describe the complete process of an exchange between two agents in detail. The process is also shown in Figure **??**. Thus, *A* initiates the exchange by sending *A*'s complete chain to *B*, together with a block index for each exchange block. *B* can perform simple checks on the chain, for example whether it contains the correct number of exchange blocks according to the exchange policy and whether all blocks are correctly signed and chained. From the chain and the exchange block indexes, *B* is able to reconstruct the complete database index of *A*.

Given *A*'s database index, *B* can calculate the difference between their databases. *B* will then request from *A* the blocks that *A* has but *B* does not. Once *A* replies, *B* has all information that *A* has. *A*'s subjective network state has become transparent to *B*.

B is then able to perform the audit of *A*. If *A*'s behavior as recorded by the chain is correct, *B* sends his chain, exchange indexes and blocks to *A*. Note that *B* already knows which blocks *A* is missing with respect to *B* and can therefore directly send them.

Now *A* is able to perform the same checks as *B*. If *B*'s data also checks out, *A* will create a new exchange block. The exchange block contains the root hash of the blocks that *A* uploaded to *B* and downloaded from *B*. As *B* also knows which blocks he received from *A* and sent to *A*, *B* should be able to calculate the same hashes. After both parties sign the block, they add them to their database of blocks. Also they create an entry for the exchange block in the exchange index map, which maps each exchange block to the index of exchanged blocks. At this point they have concluded the exchange according to the Network-State-Exchange policy.

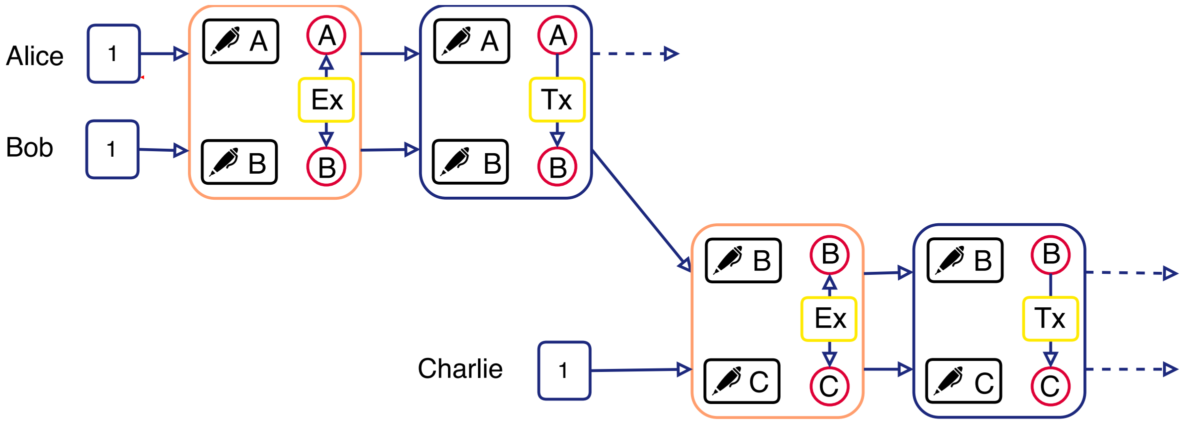


Figure 5.5: Example of three agents interacting

5.2.7. Consistency and Scalability consideration

For each interaction the Network-State-Exchange policy requires a node A to obtain all the blocks that their partner B is aware of. If that B is honest, this includes also all the knowledge of all of B's peers. Consequently each agent will approximate the full network state. This requires the storage of all blocks on the network. We have discussed this more in Section **??**.

The mechanism therefore disseminates updates to all nodes which leads to consistency and enables indirect reciprocity because all agents can have a good estimate of each others reputation and trust.

Although each exchange and transaction still only requires two agents to communicate as in the example given in Section **??** storing all interactions of the network will in practice put a bound on the scalability of the approach. In Section **??** we explore different exchange policies that might allow better scalability at lower storage cost.

5.3. Example

We now consider an example to shed more light of the exchange and interaction process. 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.

Alice creates an exchange block which includes in the field `exchange_up` the hash of Alice's genesis block and in the field `exchange_down` the hash of Bob's genesis block. Alice signs that block and sends it to Bob. Bob verifies that the hashes actually are correct. If Bob agrees, he also signs the block and returns it to Alice. Both Bob and Alice now store the block with both signatures in their database of blocks. Alice also stores a block index for the created exchange block which includes as entry only Bob's block with sequence number 1. Similarly, Bob documents that he received Alice's first block in the new exchange block.

At this point both agents are sure that they are honest and are able to interact in the application context. For example, Alice could now stream a video from Bob in Tribler. After the transaction in the application context, Alice creates and signs a transaction block and sends it to Bob who replies in a similar fashion if he agrees. Figure **??** shows the chains of Alice and Bob after the complete interaction.

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, Charlie, whom Bob would like to interact with after the previous interactions. Again, Charlie 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 the index that documents the acquisition of Alice's first block.

Charlie sees that Bob has an exchange block with Alice before have a transaction with her, which is correct according to the exchange policy. Next Charlie checks the signatures and hashes of Bob's chain. After those checks pass, Charlie tries to calculate the hashes for Bob's exchange block but realizes that he does not have Alice's first block. Therefore he requests it from Bob. After receiving it the check should pass.

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 **??** shows the blocks that each agent has after the first and second round. After the second round, Charlie 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	Charlie
Before first round	A: [1]	B: [1]	C: [1]
After first round	A: [1,2,3] B: [1,2,3]	A: [1,2,3] B: [1,2,3]	C: [1]
After second round	A: [1,2,3] B: [1,2,3]	A: [1,2,3] B: [1,2,3,4,5] C: [1,2,3]	A: [1,2,3] B: [1,2,3,4,5] C: [1,2,3]

5.4. Other exchange policies

The architecture that we have presented in the previous section is very flexible. Any size of information exchange between two parties can be recorded. It therefore allows for the implementation of any exchange policy. We described in the previous section how the Network-State-Exchange policy can be implemented and that it can give strong guarantees for security but also induces the highest cost on storage and bandwidth capacity. Depending on the application this might be necessary however TrustChain was designed to be as scalable as possible without global consensus. Therefore also less demanding exchange policies can be implemented.

In our model we describe the exchange-history policy. We show that it is enough to detect forks and honest agents can still ensure that their exchanges are correct. However, no previous exchanges can be verified as with the audit protocol. This policy therefore trades some consistency and correctness guarantees for less storage and bandwidth requirements. Similarly other policies can be explored.

Instead of enforcing a certain policy, the exchange records can also be used to create a system level notion of trust. Instead of verifying every agent before every interaction, each successful verification of an agent can lead to trust between the two. This would create a reputation system based on the adherence to the system-level requirements of good behavior.

5.5. Chapter conclusion

We have defined an extension to the original TrustChain architecture which allows for the exchange of blocks. We have shown how this architecture can be used to exchange data and enforce an exchange policy. Also, we have defined an audit protocol which allows for the complete verification of agents data.

The exchange policy ensure that data is disseminated while a strong policy like the Network-State-Exchange policy allow for strong consistency guarantees. We have theoretically established this in the previous chapter as well. Therefore this architecture is very much useful for a trust system as we envision it. On the other hand, scalability will become an issue as verification will become more elaborate with increasing chain lengths and storage requirements grow.

An alternative to enforcing an exchange policy can be to define a system-level trust system such that verifications can happen less frequently and verified partner become trusted peers. This way agents can interact with any other verified partners of their trusted peers without redoing the verification. We consider this a possibility for future research.

6 Experimental analysis

We have shown that exchange transparency and the enforcement of exchange policies offer great potential to defend against manipulators and free-riders. In this chapter we establish how a practical system responds to manipulation attempts. More specifically we emulate honest agents, manipulators and free-riders 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 free-riding 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. Implementation

The experiments are run on a standalone implementation of TrustChain with the mentioned extension and exchange 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.

An overview of the architecture is given by the drawing in Figure **??**.

We run the experiments on a local machine. In order to emulate a distributed system, each agent runs in a separate process without any shared memory with other agent processes. Each agent runs their own database instance which stores the blocks.

Agents communicate with each other through a communication interface. The interface provides an outgoing and incoming tcp port, implemented using the `zeromq` library. Messages sent through the network as defined in the Google `protobuf` format.

The main logic of the agents is implemented in message handlers. Each message has a certain type, for example a message to initiate an exchange or a message to reply to a block request. While all honest agents are handling messages in the same way, dishonest agents overwrite some correct message handlers with manipulated message handlers.

Certain message handlers create blocks. The basic TrustChain block class and cryptography functions are used from the `py-ipv8` project. The blocks are stored in a MySQL database instance. The database interface also is used 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.

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

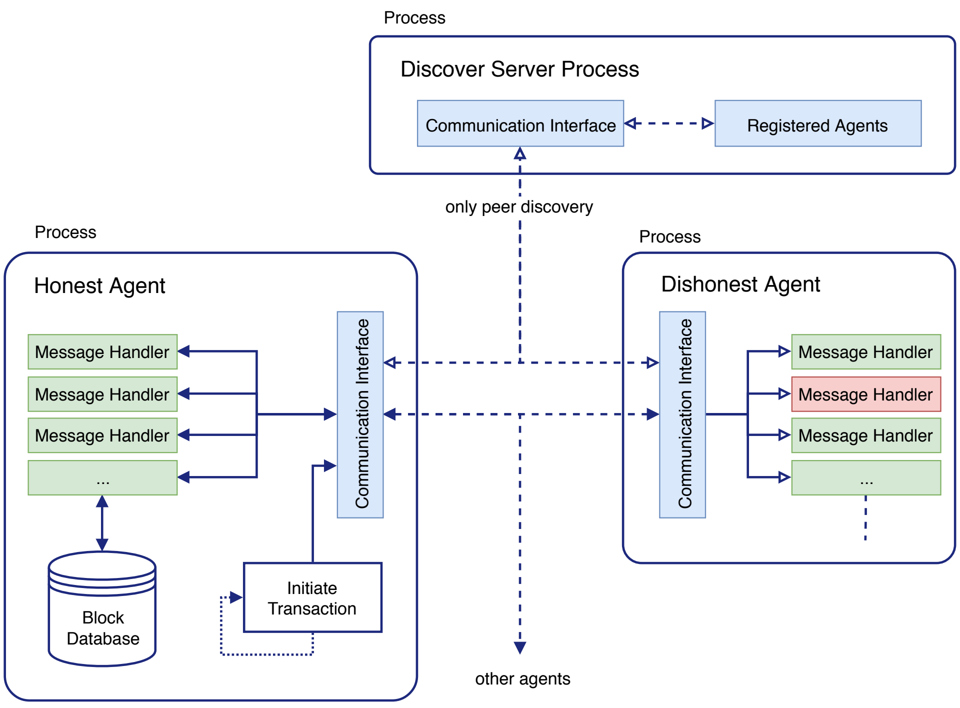


Figure 6.1: Experiment setup and software architecture. Each agent runs in a separate process. The main logic is inside the message handlers.

6.2. Experiment design

The goal of the experiments is to confirm our theoretical analysis of exchange policies from Chapter **??** and our architecture in Chapter **??** in an experimental setup. The following behaviors of agents will be analyzed in the experiments.

The honest agent without auditing only performs verifications of their direct exchanges. They do not verify the complete history of their peers using the audit protocol. For some advanced attacks this is necessary. We aim to show what additional verifications through the audit protocol can provide.

Further we are interested in the ability to exclude free-riders that either do not exchange information or do not verify them. One way to ensure the consistency of the data in the network is to incentivize agents to share data and verify data. If all agents adhere to a certain exchange policy this can be ensured. We analyze how honest agents react to agents that attempt to free-ride.

Finally we study two types of manipulating agents. A more complete list of manipulators is discussed in

Table 6.1: Agent types used in the experiments

Type	**Sub-type**	**Behavior**
Honest agents	without auditing (default)	Exchanges and verifies agents normally
	with auditing	additionally replays the history of a peer to find any deviating behavior in the past
Exchange free-rider	-	Creates exchange blocks with empty exchanges
Verification free-rider	-	Acts honestly but blindly trusts all partners without verifying their data or behavior
Manipulator	Block withholder	Creates a normal transaction and tries to hide it afterwards
	Double-spender	Creates two conflicting transactions and shares them with two different peers

Chapter **??**. In order to show that the mechanisms ensure the detection of manipulation we give two examples of manipulators: a block withholder, that is an agent who tries not share the complete chain, and a double spender, who forks his chain such that two transactions can be conducted while only recording one of them on the chain. If manipulation cannot be detected and dealt with this endangers the validity of trust and reputation calculations because the underlying data could be invalid. We show that our system successfully detects and deals with manipulators.

The experiments are designed as follows.

- **Network.** We emulate small networks of up to 6 agents, consisting of mixed sets of honest, free-riding and malicious agents.
- **Peer discovery.** We assume that all agents are connected. This is implemented with the discovery server as described above.
- **Exchanges and interactions.** All agents are willing to interact. The initiation of an interaction is probabilistic with a expected frequency of 1 transaction every 5 seconds for each agent.
- **Partner selection.** The partner selections is random with a uniform distribution. If a exchange with the selected agent is already ongoing, no new interaction is initiated.
- **Handling of malicious agents and free-riders.** Any wrong behavior that is detected through a verification will be considered fraud.
- **No forgiveness.** Honest agents do not forgive any wrong behavior. Once an honest agent detects a free-rider or malicious agent they will ignore any incoming requests and will not request interactions with them.
- **No additional communication.** Agents only communicate to exchange blocks or create transaction blocks. No information is shared on known malicious agents.

6.3. Experiment results

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

6.3.1. Dissemination free-riders

Our first experiment analysis the performance of our exchange mechanism against free-riders. The exchange of information is vital against the double spend attack and other inconsistencies. An a agent that does not exchange data is not able to provide any verification service to the network. If this is a valid strategy more agents can free-ride, thus jeopardizing the network's security. It is one of our goals to stop free-riding behavior by creating a strong incentive to exchange. In order to secure the system, honest agents should be able to detect free-riders and ignore them. This will remove any possible future profit from free-riders which should incentivize them to become honest.

In this first set of experiments we observe the behavior of three honest agents in the presence of a exchange free-rider that aims to create exchange blocks with empty exchanges. Therefore if an honest agent sends data the agent acts as if no data was received and puts an empty hash into the exchange blocks. At the same time the free-rider does not request any blocks from the partner.

Figure **??** shows the number of transactions of each agent against the time of the experiment. All honest agents steadily increase their successful interactions throughout the experiment and end up between 40 to 60 transactions. The free-riding agent is not able to perform a single interaction. That means all honest agents detected the free-rider as a dishonest agent and consequently ignored him.

As expected, an agent has no chance to interact with the honest agents without a complete exchange. We have shown this property in Theorem **??**.

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.

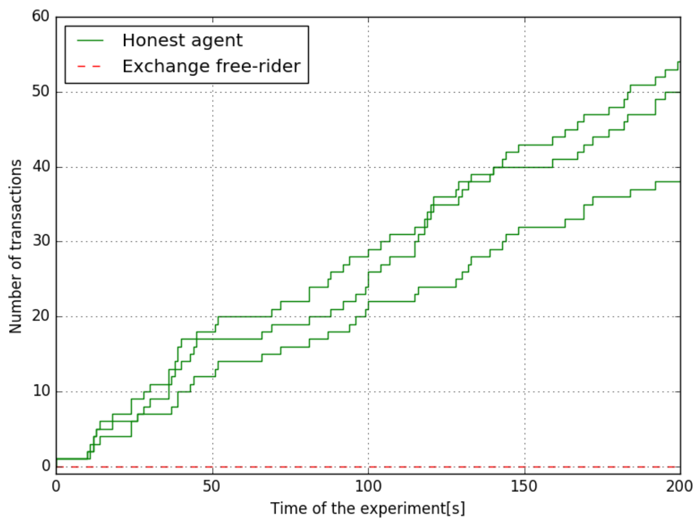


Figure 6.2: Transactions over time of honest agents with exchange free-rider (both types give same result)

6.3.2. Colluding free-riders

Our second experiment analyses the impact of multiple free-riders in a collusion attack. In order to be accepted as honest agents without gossiping, colluding agents can work together and sign empty exchanges for each other. This will create a seemingly valid history for the free-riders. Their goal is to interact with each other *and* with the honest agents while exchanging less than necessary. Honest agents should ignore them in order to keep their network safe and honest.

In order to analyze this situation we ran an experiment with three honest agents and three colluding gossip free-riders. The transactions are plotted against time in Figure **??**.

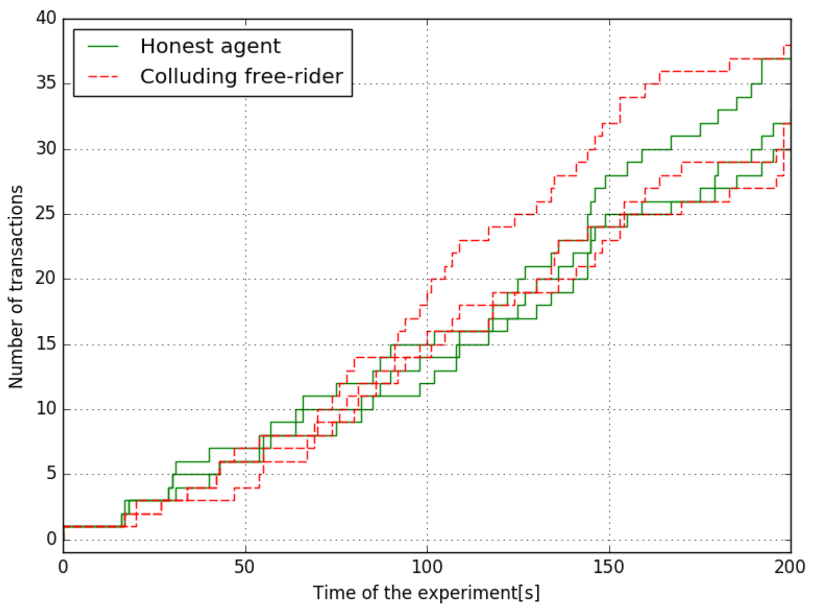


Figure 6.3: Transaction history of three honest agents and three dissemination free-riders that are cooperating

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. However the plot does not show which agents interact with each other. Therefore an interaction matrix is shown in Figure **??** which shows for each agent, how many interactions they had with each of their peers.

From Figure **??** it becomes clear that no interactions (red squares) happen between honest and dishonest

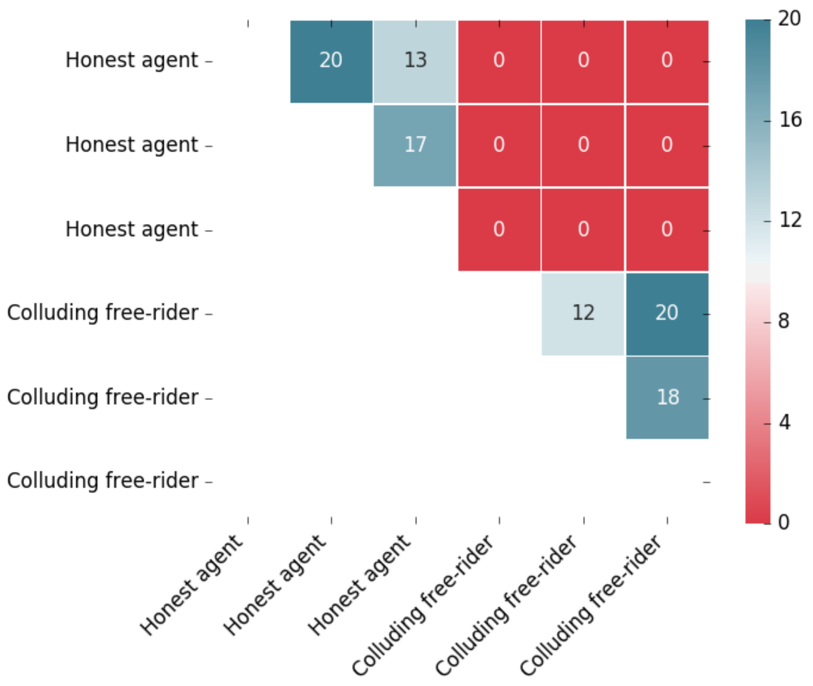


Figure 6.4: Interaction matrix of quantifying the network separation

agents. Honest agents interact with honest agents, while dishonest agents interact with dishonest agents. When dishonest agents try to interact with honest agents, honest agents will not sign an exchange block with empty hashes. So even though their history is correct according to the exchange policy, honest agents are still able to separate dishonest from honest agents. This leads to network separation.

6.3.3. Malicious behavior

In the next experiment two types of malicious behaviors are analyzed: forking and block withholding. Similar to the free-riding, honest agents who use the TrustChain architecture and our exchange and verification mechanism should isolate the malicious agents. We have discussed that manipulation of the data the underlies reputation and trust calculation will invalidate the trust and reputation built in the system. It is therefore key to defend against manipulations. The results of the experiments are presented in a similar format as previously in Figure **??**.

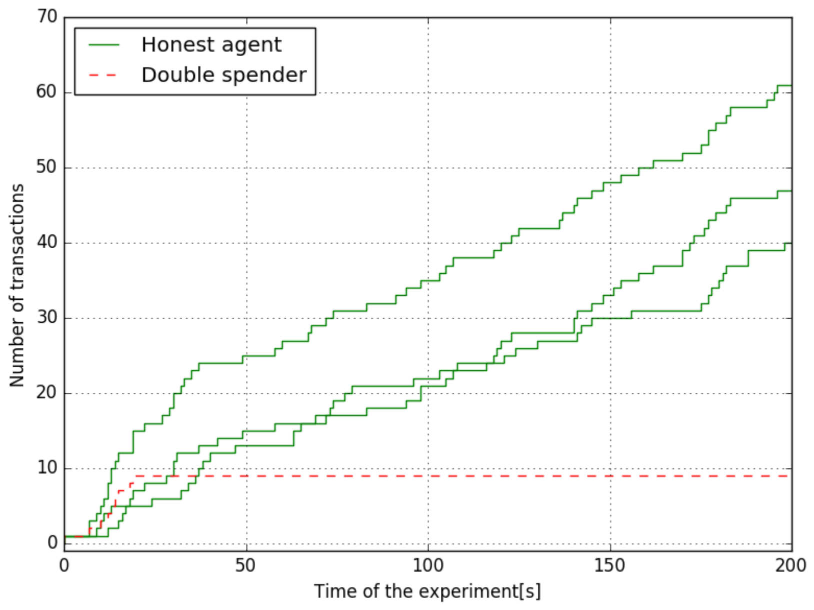
Both agents perform an attack with a 25% chance in each round. The transaction hider is able to obtain six successful interactions before all honest agents ignore him and the line flattens. When trying to withhold a block any honest agent is able to detect the fraud. The agent is not able to hide any 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 same can be observed for the double-spender. 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 and the network size, this can take several rounds. Therefore the forking agent is able to perform 9 interactions in the first 20 seconds of the experiment. After those interactions the red line indicating the forking agent flattens, meaning no more successful interactions happen.

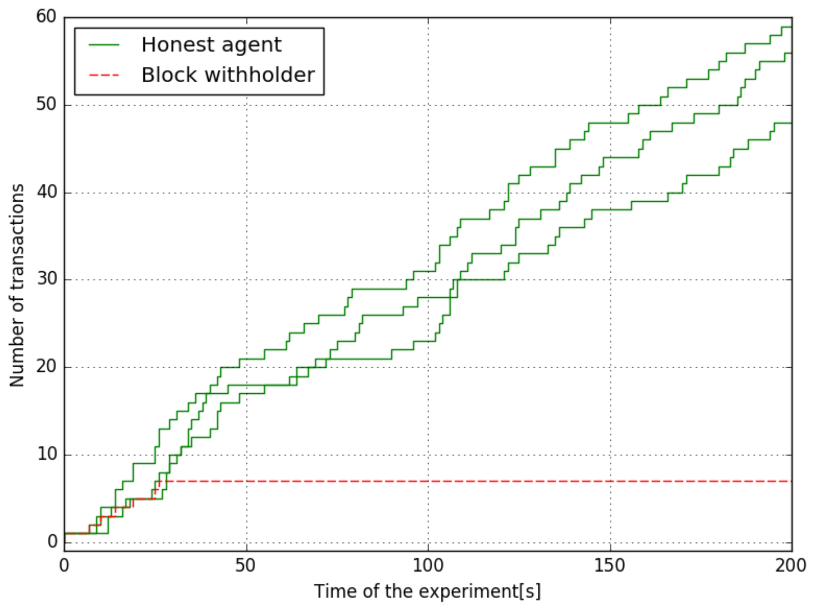
6.3.4. Verification free-rider

In the previous experiment we showed that malicious agents can be detected and isolated. In Chapter **??** we have presented a theoretical study that shows that with the Network-State-Exchange policy also those agents that knowingly interact with those malicious agents can be isolated. This is possible *auditing* the complete history of an agent and redoing the verifications. Any honest agent that obtains the full knowledge of the subject can perform this check ad find any failing verification which signals dishonest behavior.

In this experiment we study an combined attack. On the one hand a double-spender is creating a fork



(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 6.5: Experiments of honest agents with sinlge malicious agents

in order to gain an advantage. On the other hand another agent does not perform any verifications and interacts even when possibly knowing about a double spender. This is similar to the verification free-rider and the double-spender colluding. The attack is difficult to detect because the agent that does not perform verification acts completely honestly in all other regards.

We study two situations how honest agents behave in this situation, once without auditing, once with auditing.

The results of the first case are presented in Figure **??**. 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 **??** shows that most of the interactions of the forking agent come from the verification free-rider who does not perform any checks and therefore does not care about the fork.

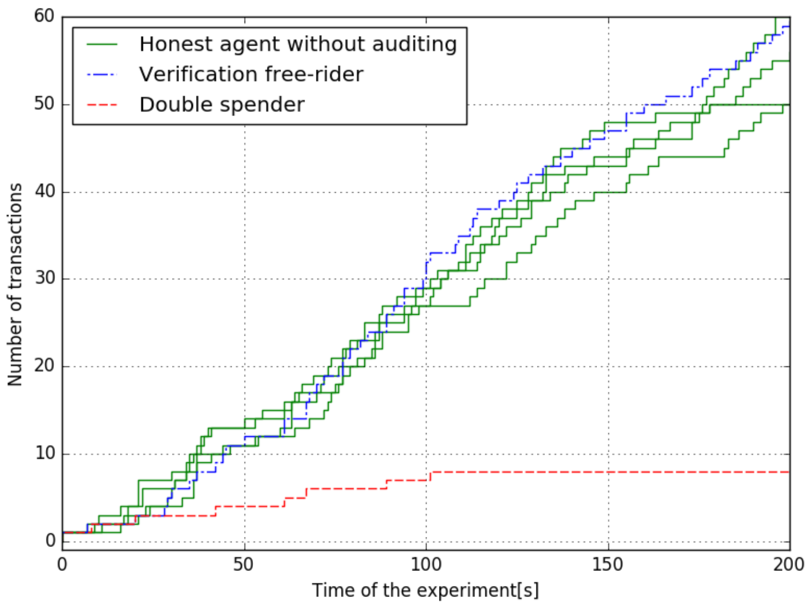
Next we repeated the experiment, but this time the honest agents perform auditing of their partners before every interaction. The results are shown in Figure **??**.

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 100 seconds. It is quite obvious, at least from the 100 second mark onwards, that the blue and red curve are running exactly in parallel. This is because after detecting the forking agent and the verification free-rider, the honest agents are able to identify *both* 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.

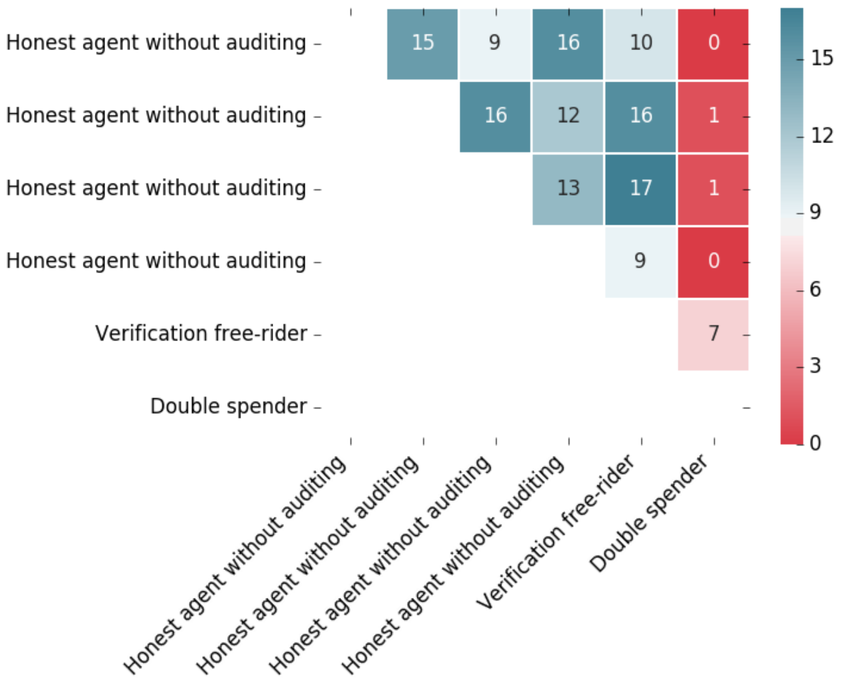
6.4. Chapter conclusion

In this chapter we have studied the recording of exchanges and implementation of Network-State-Exchange policy in an experimental setting. We have shown that any honest agent is able to ignore agents that do not exchange correct data. Even if agents are working together they are not able to interact with honest agents without exchanging the correct data and creating signed proof of that exchange.

Even any previous collusion with a malicious agent can be found through proof-of-knowledge of a double spend on the verification free-rider's chain. We can conclude that the mechanism of recording block exchanges enforces agents to obtain information and ignore any known malicious agents or become a fraud themselves.

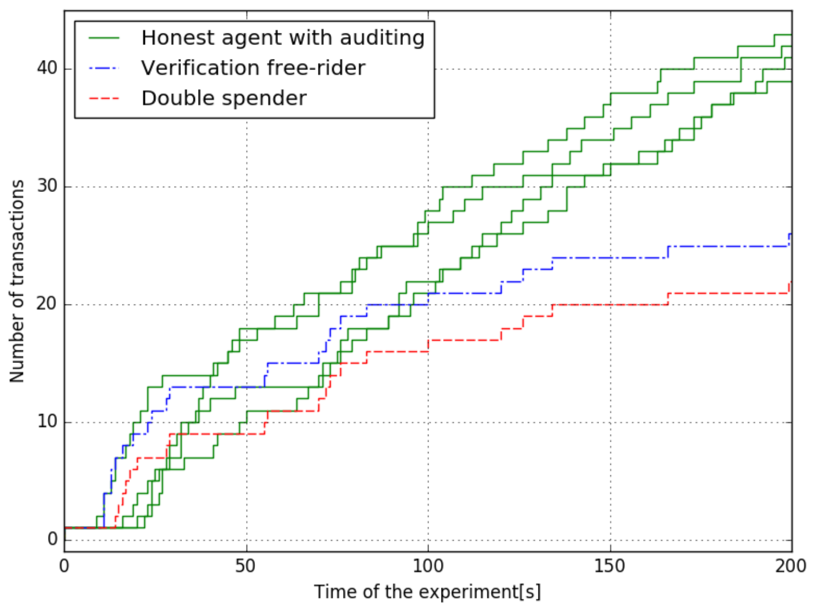


(a) Transaction history of three honest agents interacting with one strategic manipulator who performs a fork and a verification free-rider

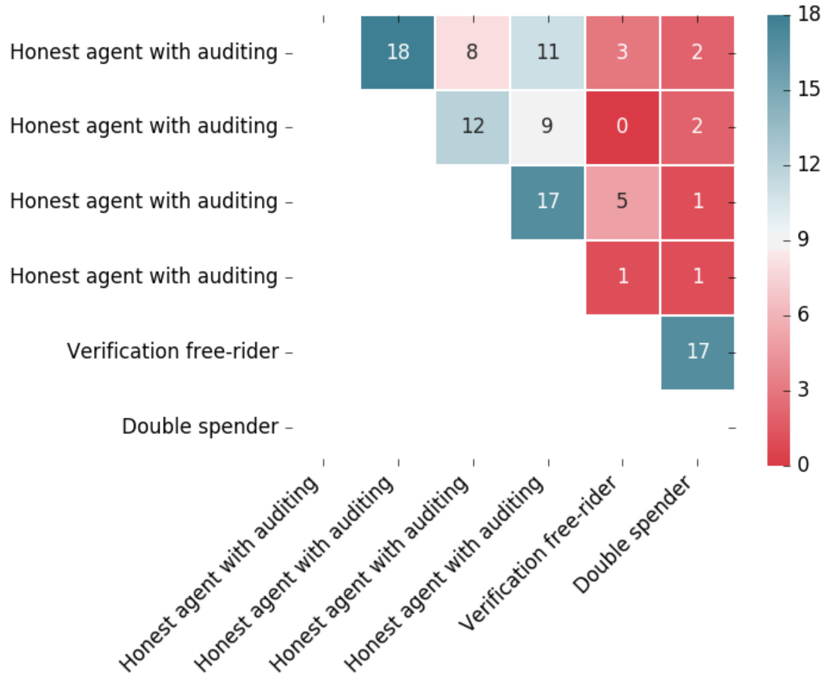


(b) Interaction matrix of three honest agents with one strategic manipulator who performs a fork and a verification free-rider

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



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

7 Conclusion

Creating trust on the internet is a challenging task. Many of the current centralized reputation systems will, in the long run, not be able to satisfy the demands for privacy, security and platform freedom. A distributed trust system is a possible solution, yet it bears its own challenges. In this work we addressed the problem of ensuring the dissemination and verification of records of interactions, which are the basis of any reputation or trust system. We will discuss in this last chapter our contribution to the design of a trust system and the future research towards a fully distributed, scalable and secure global trust system.

7.1. Contributions

Evidence of encounters are the core of any trust and reputation system. A reputation is little more than a summary of a persons history of interactions in a certain context. Many positive interactions lead to a good reputation whereas negative interactions lead to a bad reputation. The better someone's reputation, the more we trust them and are willing to reciprocate their positive interactions. In a digital trust system the records of interactions therefore play a critical role in ensuring the correct value and validity of trust. Yet, in a distributed system without central government, no single point in the network has all information. A key problem to solve is there to ensure the correctness of interaction records and their dissemination in the network.

7.1.1. Model

We first designed a model to study the problem theoretically. We model a network of agents. Agents can perform interactions and exchanges of information. An exchange can contain information about other interactions as well as other exchanges. We showed that without exchanging information, the network of agents is not able to defend against double spending attacks. In order to ensure that agents do gather and disseminate information, each exchange of information needs to be become part of an agents history similar to interactions. Agents thus need to prove with their history that they did exchange information. It is thus possible to ensure a certain exchange policy. We show that if agents are bound to a policy to exchange all their information with every encounter, additional desired security properties can be ensured. This includes that agents can verify the past behavior of their partner and thus be sure they did not perform any invalid interactions.

7.1.2. Architecture

After proving theoretically the importance of exchanging information, we designed an architecture that allows to record and verify transactions and exchanges of information. For that purpose we extend the TrustChain fabric, which is a blockchain-based, scalability-focussed solution for recording interactions in distributed systems. The entangled hashchain approach of TrustChain creates advanced tools for the detection of manipulation attacks. However TrustChain's security greatly relies on the willingness of agents to obtain their peers blocks and verify them.

Our extension adds exchange blocks to TrustChain which record any block exchanges between agents. Exchange blocks contain a hash of the blocks that were exchanged and they are recorded on the chains in the same way as normal transactions. This creates for each agent a tamper-evident history of exchanges and transactions. By recording a signed hash of each exchange, agents can be asked to provide the exchanged blocks such that the hash can be recomputed by any verifier. Failure to provide the correct blocks will lead to mistrust. This allows honest agents to distinguish between honest agents that also exchange data, free-riders that do not exchange and manipulators that have invalid data on their chain. In applications that require the highest level of security, agents can be required to exchange all knowledge. This allows any agent to redo any verification in the subject's history. If all verifications pass, the agent acted exactly like the verifier would do and thus can be considered very trustworthy.

7.1.3. Implementation

Our final contribution is an implementation of the architecture and verification mechanism as well as an experimental analysis of its manipulation resistance. In our experiments small groups of agents are emulated. The groups consist of honest agents that implement the exchange and verification mechanism according to design, as well as free-riders and manipulators. We show with two examples that honest agents are able to detect manipulating agents. But it was not possible yet to also defend against lazy agents that do not help in detecting those agents. Our extended architecture allows honest agents to detect exchange free-riders, even if they are colluding and also agents that knowingly interact with malicious nodes. As a consequence, any agent that wants to interact with an honest agent in the future needs to exchange data with their partners and make sure that they are not in any way malicious.

We conclude that exchange transparency ensures that agents acquire and distribute knowledge. Our extension of TrustChain is flexible to allow different levels of information dissemination. It allows agents to replay the history of another agent to validate their complete historic behavior. As such our work is a major step in securing our future distributed trust system.

7.1.4. Research question

The research question we set out to answer is:

How to ensure that reports of encounters are honest, consistent and disseminated in a distributed trust system?

We can make the following conclusions. Recording exchanges leads to a strong incentive to disseminate and verify encounters between agents. This way honesty of encounter reports can be ensured as well as consistency. Therefore our architecture could be seen as a possible solution to problem of recording encounters in a trust system.

However, our system creates restrictions for the scalability of the system. We require much more storage than only the own transactions. Also, the audit verification can become very costly in terms of computing resources once agents' chains become significant in size. Therefore, this system is only partly applicable in the context of a global-scale trust system.

7.2. Future work

Looking ahead, there are still major challenges to achieve our ambitious goal of building the internet of trust. On the one hand, we have created a prototype for exchange transparency but some challenges remain and need to be researched. On the other hand multiple other large challenges need to be solved like a strong identity system and the Sybil attack.

7.2.1. Further development of exchange mechanism

The recording of exchanges has great influence on many components of the trust system. Not all of these could be analyzed in this work.

First of all a larger study on the scalability of our mechanism needs to be done. Redoing the verification of an agent's complete history is very work intensive once the history grows. It is possible to do this in multiple steps, each time two agents interact they only need to verify the history that has changed since their last encounter.

Also, the Network-State-Exchange policy will require each agent to store all data of the network. This is, in many cases, not be acceptable. Instead, other less demanding exchange policies could be put in place. This is a trade of between consistency and scalability. Our architecture is flexible to allow different policies to be applied.

Another promising direction for further research is building a system-level trust system which we mentioned in Section **??**. Instead of enforcing a certain policy it is optional exchange and verify data. However, each exchange and verification will lead to a more positive reputation of that agent on a system-level. Agents that exchange, store and verify many blocks should be rewarded with better conditions. An open question will be how to combine the system-level with application-level reputation. And whether the incentive can be strong enough to ensure proper consistency and dissemination.

7.2.2. Future development of trust system

In Chapter **??** we have introduced the architecture of a trust system. We have addressed the mechanism for recording transactions and the distribution of those records. But other challenges remain.

A strong identity system which relates the public keys to true identities through government issued documents can greatly improve security. As we have discussed in Section **??** some attacks like whitewashing and Sybil attacks are possible because identities are cheap. If on the other hand identities are coupled with real-world, single instance values or documents such as passports, it becomes much harder to renew an identity. A downside of such an approach is that it creates a dependency on external documents. In countries where governments are corrupt or unable to act such documents might not be obtainable. In that case biometric data could help to create strong identities.

Another way we envision to create a stronger protection for our trust system is to use latency. The connection latency between nodes in a network is a physical value that defines a neighborhood for each node. If we make trust dependent on a low latency we believe that attacks will be much harder. An attack usually requires an interaction, so some initial trust. However trust can only be obtained from inside a physical distance. Thus attackers need to be able to access machines that are close to their targets. Attacks from distant governments or hacker groups require infiltration of local machines, putting another obstacle in their way.