Trust in Distributed Systems

Bram van den Heuvel Yinghao Dai

May 17, 2018

Preface

This document is a continuous work in progress while the Delft University of Technology Blockchain Lab researches cooperation in distributed systems. Its aim is to support members of the lab by

- providing a convenient introduction to (the mathematics of) the subject,
- supplying an overview of previous work by both the lab and others,
- making explicit the gap between mathematical theory and real-world deployable systems.

In this way, we hope the lab can work in the most structured way towards its long-term goal of realizing a universal mechanism to create trust.

Contents

Chapter 1

Introduction

1.1 Historical Perspective

Cooperation is a hard problem. In many environments, self-interested and rational choice making results in a suboptimal outcome for all. Arguably the best known example of this problem is the prisoner's dilemma, where defection is a dominant strategy. The prisoners' inability to coordinate and build expectations of the choice made by the other prisoner ensures that both will be worse off than if they had collaborated.

In real life, we encounter more often a repeated, or iterated version of the prisoner's dilemma. We often meet the same people. For this game, different strategies exist.

1.1.1 Trust

There are various definitions of trust, but all have a few basics facets to them [1]:

- one party (trustor) is willing to rely on the actions of another party (trustee); the situation is directed to the future
- the trustor (voluntarily or forcedly) abandons control over the actions performed by the trustee
- the trustor is uncertain about the outcome of the other's actions; they can only develop and evaluate expectations
- the uncertainty involves the risk of failure or harm to the trustor if the trustee will not behave as desired

We can view trust as necessary catalyst for practical collaboration. Trust systems have existed ever since people lived in close-knit, local societies. Mutual dependence for survival, a lack of movement and gossip were sufficient to build small systems of trust. In the industrial age, large companies and brands became commonplace. The large investment in brand recognition is can be viewed as a "costly-to-fake" signal, leading consumers to trust in the companies intention to sustain its brand value, and sell high-quality products.

More recently, a new type of trust system emerged. Companies like eBay and AirBnB allow consumers to interact directly without having prior knowledge of each other. A "two-way ranking" system is facilitated by these companies, where any two consumers interacting via the

platform give a review of their transaction partner. These companies then capitalize on the trust that consumers have in each other.

We want to realize trust systems which allow individuals which have never interacted before, to trust each other without any central entity involved. Blockchains are the first successful attempt at realizing such a system.

1.2 Blockchains

Blockchains are data structures which utilize cryptographic primitives such as public-key cryptography and trapdoor functions. The most well-known blockchain is arguably Bitcoin, the first implementation of a blockchain used for financial transactions. The currency uses a blockchain as a ledger to store past transactions. These transactions are grouped in so-called "blocks". The ledger is composed of a sequence such blocks; they form the ledger. All blocks point to the previous one by containing the a cryptographic hash of the previous block. Blocks can only be appended to the chain, and are immutable; modification of a block changes its hash and in this way "breaks" the chain.

1.2.1 Consensus in proof-of-work blockchains

All major blockchains currently use "proof-of-work" as a fraud prevention mechanism. New blocks are formed by "miners", nodes in the network which collect transactions and group them with the hash of the previous block. They repeatedly append a random nonce to the data and calculate the cryptographic hash, until they find a nonce which results in a cryptographic hash satisfying certain properties. Once such a nonce is found, the transactions, the previous block's hash and the nonce are published to the network; they form the new block.

The miner which "finds" a new block is awarded with some currency in the ledger. Sometimes, two blocks are found closely after one another. Then, the longest chain determines the latest state of the network. The proof-of-work mechanism ensures that mining is costly, and that no single entity can control which chain is the longest by producing many blocks.

In all major blockchains, consensus on the latest state of the ledger is formed by mining a block. This requires that all transactions are globally known, which fundamentally limits the scalability of such a system. In pursuing a more scalable solution, another data structure has been invented called "TrustChain".

1.3 TrustChain

In TrustChain, all network participants have and maintain their own chain. There is no mining, and no global consensus. TrustChain works fundamentally different than existing blockchains; we sometimes call these blockchains "fourth-generation blockchains".

Users create a new chain by generating a public-private keypair and a first, empty block. When they participate in a transaction with another TrustChain user, they create a block pointing to both their own, and their transaction partner's last block. Next to the transaction contents, they append a signature using they private key. An example of blocks and references is given in Figure 1.1.

Users share (part of) their chains with the people they transact with, and possibly share their blocks with and collect blocks from others. This data structure is extremely scalable. But because there is no global consensus, several problems arise.

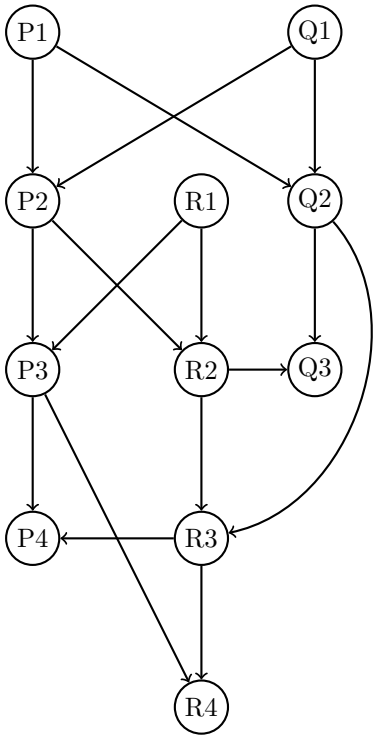

Figure 1.1: An example of a collection of blocks and pointers. Adapted from [2].

We will first list several other requirements for such a distributed ledger. Then, we will develop a model which attempts to capture the motivations of those participating in the network. This leads us to frame the problem in a game theoretic way, after which we consider the notion of reputation.

After we then examine several ways in which reality is still different from the model proposed, a brief summary of previous systems attempting to solve the trust problem is provided. We then consider other proposals which partially solve the trust problem, after which we finally provide an overview of open problems.

Chapter 2

Requirements

2.1 Decentralization

Naturally, the system should be completely decentralized. It is especially the cooperation independent of a central party we want to realize.

2.2 Scalability

A universal mechanism to create trust requires it being usable by the entire world with absolute minimal limitations of usage. For this reason, we require of any system which attempts to solve the trust problem that it is scalable in three ways. The first being in the number of users that can participate in the system freely, concurrently and without interruption. Secondly, we need users to be able to make a large number of transactions in short periods of time, always. The third scalability concerns time: as the system gets used for a longer period of time, the previous two scalability requirements shouldn't become insurmountable to achieve.

2.3 Robust and truly peer-to-peer

All those in the network should perform the same set of tasks. The appointment of special roles often results in fragile systems which are (temporarily) dependent on a very small number of participants. We don't want a small group of nodes performing the tasks normally performed by a a central authority; instead, we want a truly distributed network in which responsibilities are shared by all. Moreover, a single node type results in simpler, more resilient designs.

2.4 No "early adopter takes all"

Many cryptocurrencies know an extremely small group of early adopters which own an large portion of all coins. We aim to create the possibility to trust strangers, rather than redistributing wealth.

Chapter 3

A rational choice theory

While having been criticized for being unsuited to model economic choices made by human beings, a neoclassical economic model of utility maximization can be well-suited to analyze a system such as the one we are interested in. Namely, in practice the making of these choices will be automated and executed by a program well able to quantify utility. Moreover, notice that the model developed below assumes only agents' ability to order different utilities rather than calculate their magnitudes in absolute terms.

3.1 Notation

We will start developing our model by considering a market $M := \{A | A \text{ is an } \textit{active} \text{ agent}\}$ where a unitary good v, which has value and can be created by doing one unit of work w, is traded. Agents are said to be active if they will trade in the future. Denote by V_A the set of all v owned or consumed by A and by W_A all units of work A has performed. All agents are able to transact in an atomic, costless manner. Transactions are ordered in time and record a single v being sold by agent A to agent B as transaction $T = (A, B)$. All past transactions are recorded in a history H, of which we will for now assume that it is complete and available to all agents in M. When a transaction T is appended to history H, we denote this by $H + T$. The total number of transactions in H is denoted by $|H|$, and all transactions are completely ordered.

All agents $A \in M$ have a reputation function $r_A : (M \times \mathcal{H}) \to \mathcal{R} = (B, H) \mapsto r_A(B|H)$, where $\mathcal{R}$ denotes the set of possible reputations and $\mathcal{H}$ denotes the set of possible histories H. We now define $r_M : (M \times \mathcal{H}) \to \mathcal{R}$ as

$$(A, H) \mapsto \max_{B \in M \setminus \{A\}} r_B(A|H),$$

the highest reputation assigned by any market member.

We denote by $u_A^R : \mathcal{R} \to \mathcal{U}$ the utility A derives from reputation $r \in \mathcal{R}$, and by $u_A^V : V \to \mathcal{U}$ the utility A derives from valuable v. We assume that a unit of work performed by A costs utility u_A^W, while a v received lets A enjoy a positive u_A^V. The total utility of agent A is now denoted by $u_A^T : \mathcal{H} \to \mathcal{U} = u_A^R + \sum_{v \in V_A} u_A^V(v) + \sum_{w \in W_A} u_A^W(w)$. $\mathcal{R}$ and $\mathcal{U}$ are totally ordered and have additive identity 0, while for u_A^R we have $x, y \in \mathcal{R} : x > y \Leftrightarrow u_A^R(x) > u_A^R(y)$. Naturally, we suppose that agents in M attempt to maximize their u^T by taking part in the right transactions.

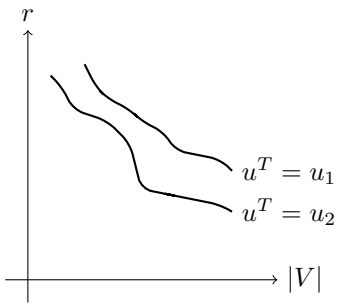

Figure 3.1: Utility curves for agents in M. We have $u_1 > u_2$.

3.2 Interpretation

In such a market M, an agent A's reputation depends on the entire history H. We can view A's reputation $r_M(A|H)$ as a measure of A's future ability to do create transactions in the network. The reputation of A will influence its ability to participate in M in the future. In this way, the utility derived from v and utility derived from A's reputation represent A's short- and long-term interests, respectively.

As is common in micro-economics, we can visualize an agent's trade-off between these short- and long-term interests. On the horizontal axis we measure $|V|$, while on the vertical axis we measure A's reputation. In the case where u_A^R and u_A^V are continuous, we can draw utility level-curves such as in figure 3.1. Note that they cannot cross.

3.2.1 Transaction condition

We are now ready to formulate a simple transaction condition for agent A. Its utility will increase by doing a transaction T if $u_A^T(H+T) > u_A^T(H)$, or

$$u_A^R(r_M(A|H+T)) + u_A^W(w) > u_A^R(r_M(A|H)) \text{ for } T = (A, B),$$
$$u_A^R(r_M(A|H+T)) + u_A^V(v) > u_A^R(r_M(A|H)) \text{ for } T = (B, A).$$

Note that this condition isn't dependent on r_A. This implies that in deciding which trades to perform, an agent's own measure of reputation is irrelevant: it only matters how an agent thinks the rest of the market perceives its actions. In this way, taking the u_A's as given, A's behavior is governed completely the r used by the $B \in M \setminus \{A\}$. Once a certain reputation function is used by a large fraction of participants, all are compelled to use it; reestablishing the status quo. For this reason, we'll from now on assume that all agents use the same reputation function r to calculate reputations from H. We'll say that r *prescribes* M.

3.2.2 Reputation

Why should agents value their reputation at all, and not have $u^R \equiv 0$? As we will show later, there needs to be some mechanism to exclude non cooperative agents. Agents in M are active and as such get value from doing transactions in M. So naturally, they get utility from these transactions. In this way, the ability to do transaction is valuable and gives positive utility.

3.3 Conditions for a system of trust

A question of great importance is which function r to use. Before we dive deeper into this question, we must ask ourselves what behaviors it should generate from agents. An entire chapter is dedicated to this problem (chapter 4). For now, we will only state a requirement that the system as a whole should, at least, achieve.

3.3.1 Total contribution and "leakage"

We first define for agents $A \in M$ the function $\sigma_A : \mathcal{H} \to \mathbb{Z} = |W_A| - |V_A|$, the net amount of work performed by A. Clearly we would like to have a small number of agents with $\sigma < 0$, but we will not pursue individual behavior further in this section. Instead, we will consider the summed net contribution of all active agents A:

$$L_M(H) := \sum_{A \in M} \sigma_A(H).$$

We call L_M the *leakage* occurring in M. It represents the work performed by active agents which is lost to agents that have become inactive. For M to function properly, we do not want L_M to become too large. As such, a desirable property of this system is that $\lim_{|H| \to \infty} L_M(H) < \infty$, implying that the amount of lost resources rapidly decreases towards zero over time.

In section 4.2, we will consider several ways in which agents can manipulate the system. We will find that some intuitive functions r do not satisfy the finite leakage property in those cases.

3.4 Comparison to a classical market

It is worth noting a fundamental difference with neoclassical micro-economic analysis of choice theory. In a typical micro-economical market, all buyers and sellers are anonymous. When a product is traded at equilibrium price, participants could first buy the product, and sell it after (or vice-versa) with no change to their money, products or utility. In contrast, consider the decision faced by an agent A in our market M. This agent doesn't pay with money, but with its reputation. As soon as A buys the product, A's reputation changes which influences its ability to sell the product after. The same applies for the reverse order of actions: as soon as A sells the product, its reputation changes influencing A's ability to buy it back later.

This change in reputation can also present itself as a different willingness by others to transact at all.

Chapter 4

Reputation

4.1 Intuition

It is quite easy to describe intuitively what we want the reputation function r to achieve. It should reward good behavior, such as contributing to those who contribute, and punish behavior that threatens the health of the system, such as free riding. The difficulty is of course how to formalize the different behaviors.

4.2 Requirements and examples

We describe several properties which the reputation function should at least respect if we want to achieve finite leakage.

4.2.1 Giving feedback

Any reputation function must be in some way recursive: it must give a high reputation to those contributing to the agents with a high reputation. If this would not be the case, a high reputation is not valuable and as such not something that provides utility.

A reputation function which does give feedback, with $\mathcal{R} \subseteq \mathbb{Z}$ and $H = H' + T$ if $|H| \geq 1$, is the following:

$$r(A|H) = \begin{cases} 0 & \text{if } H = \emptyset \\ r(A|H') + 1 & \text{if } T = (A, B), r(B|H') \geq 0 \\ r(A|H') - 1 & \text{if } T = (\cdot, A) \\ r(A|H') & \text{else.} \end{cases}$$

That is, A gains a unit of reputation for a transaction with B, provided that B has a nonnegative reputation. Being on the consuming end of a transaction costs a unit of reputation.

Because no agent will be awarded reputation for working for a free-rider (as B did in the previous example. When M is prescribed by the above r, all agents can consume without contributing only once; showing that L_M is bounded above by $|M|$.

4.2.2 Sybil resistance

Because all agents create and maintain their own chain and are allowed to join the market freely, creating a new chain (and as such, a new identity) is cheap. As such, we can't assume all agents

act independently; a single agent might create other, *sybil* identities who secretly collaborate. Their goal is to accumulate utility for some subgroup of the identities, after which the other identities are often abandoned. From this point onward, we adapt our model to reflect this possibility.

As such, we need our reputation function to be *sybil resistant*. With our previous definition, r doesn't protect against the sybil attack. An agent $A \in M$ could create an infinite number of identities $A_0, A_1, \ldots$ and create with each identity a transaction recording the transfer of value without actually performing any work.

We will now adapt our reputation function to be sybil resistant. Previously agents had some freedom in choosing who to perform work for, allowing for the possibility of working for one's sybil identities. We will now remove this opportunity by awarding a unit of reputation only when performing work for the agent in M with the highest reputation. Again, receiving work costs a unit of reputation. Assume again $\mathcal{R} \subseteq \mathbb{Z}$ and $H = H' + T$ if $|H| \geq 1$,

$$r(A|H) = \begin{cases} 0 & \text{if } H = \emptyset \\ 1 & \text{if } H = T = (A, \cdot) \\ r(A|H') + 1 & \text{if } T = (A, B), B = \operatorname{argmax}_X r(X|H') \\ r(A|H') - 1 & \text{if } T = (\cdot, A) \\ r(A|H') & \text{else.} \end{cases}$$

Let's start our analysis of this reputation function by considering the first transaction $T_1 = (A, B)$ happening in M. After this transaction, we have $r(A|T_1) = 1$ and $r(B|T_1) = -1$. Under our adapted model, B has the possibility to abandon his previous identity and chain. B's alternative is to start anew with an empty chain and identity B' without history, and as such with reputation $r(B'|T_1) = 0$. Clearly, B prefers reputation 0 over reputation -1 and will create such a new identity B'. Because at this point, B will have become inactive, we now have $B \notin M$. Moreover, $\sigma_A = 1$ and as such $L_M(H) = 1$.

Because work performed will only be awarded reputation if it is performed for the agent with the highest reputation, this will be the only work performed. In this way, every transaction will remove a unit of reputation from agent with the highest reputation, repeatedly changing which agent has the highest reputation $r = 1$, all others having $r = 0$. We conclude that for this definition of r we have $\lim_{|H|\to\infty} L_M(H) = 1$.

4.3 Deciding on fairness

Possibly, there are many possible reputation functions which achieve finite leakage. Because the chosen reputation function prescribes which transactions will take place and which history will form, the influence of this choice couldn't be greater. As such, it is important to consider what values one would like to capture in this function. Which behaviors should be rewarded? The reputation function discussed previously strictly requires contribution before consumption, for example.

Other reputation functions could possibly give those with a certain reputation more leverage or market power than others, for example. Other possibilities include advantages for those with long histories (early adopters), or advantages for those with an ability to invest outside resources. There is also the possibility that some reputation functions will result in isolated communities doing transactions among each other. One can also wonder whether this is desirable.

Chapter 5

Reputation in practice

5.1 Introduction

The model used in the previous chapters is in several ways quite heavily restricted. This chapter lists several ways in which the previous model fails to accurately capture reality. The simplifications made are often large enough to make known results from game theory unusable. As such, it is primarily this chapter which captures the difficulty of designing scalable reputation systems useful in the real world. Not all of these differences make the this task harder, though.

From this chapter onwards, more terminology will be used to describe the problem from a computer science perspective, rather than a game theoretic one.

5.2 Knowledge of past interactions

5.2.1 Ordering

The analysis in chapter 4 relied heavily on a complete ordering of transactions. In reality, however, scalability requires that transactions can be created independently of each other, at the same time. As such, in reality we only have a partial ordering. This makes any reputation functions defined previously unusable in practice. See [2] for a model which incorporates this requirement.

5.2.2 Sharing

In TrustChain, participants should store at least the transactions they themselves were involved in. Because transactions are, shortly after creation, known only to the authors, they should share the transaction made with other active agents. They can either broadcast the transaction proactively, or send it when requested by others.

5.2.3 Limited storage

Whereas the previous chapters assumed all historic events are known by all at all times, this is clearly not the case in practice. No ledger can be duplicated completely for all of its participants and scale well at the same time.

Agents must therefor decide selectively what to store. This could, for example, be a random sample of past transactions, or only some aggregate statistics. Other possibilities are to store only recently created transactions, or to compress historic data in some lossy way.

5.2.4 Limited networking capabilities

It is not only limited, costly storage which prevents this from being a possibility; it is also the limited bandwidth of computer networks which ensure that such an amount of data cannot be communicated.

5.2.5 Limited computation power

Where fundamental game theory often rests upon an assumption of infinite computational abilities, they are very much finite in practice.[1] These limitations become apparent in algorithms such as NetFlow [2].

5.3 The human factor

Thus far we implicitly assumed that all agent's actions would be determined by automated decision making. The system should definitely be "incentive-proof" to a standard removing the opportunity for exploitation using automated decision making. In practice, however, many agents are human beings or are software scripts being run by human beings which don't modify default behavior and in this way, act honestly and cooperatively.

5.3.1 Power of defaults

As mentioned above: a significant percentage of users do not modify default software installations. Estimating the size of this fraction is hard, however, and making assumptions about it is seen by some as academically impure.

5.3.2 Norms

Other things that set human beings apart from automated decision making are values, norms, feelings of responsibility, higher goals and a sense of belonging. All these phenomena provide significant, alternative means to compel people to cooperate.

5.4 Networking

Besides limited bandwidth, overhead and transmission errors, there are more difficulties caused by the limitations of computer networks.

5.4.1 Availability

Other agents are not always available for communication. This may impact an agent's ability to request certain blocks. Otherwise, an unexpected network failure might result in a complete interruption of communication during the creation of a transaction.

[1] *Did you know that Bitcoin mining is NP-hard?* `https://freedom-to-tinker.com/2014/10/27/bitcoin-mining-is-np-hard/`

5.4.2 Latency

Fundamental physical limits ensure that latency will always be above a certain threshold, once physical distance is sufficiently large. Otherwise, a connection between two agents might have an above average amounts of hops in it resulting in higher latency through packet switching.

Chapter 6

Previous Systems

The systems mentioned in this chapter were notable milestones in the search for free-rider prevention mechanisms. They are not implemented in Tribler.

6.1 EigenTrust (2004)

EigenTrust calculates trust in a peer-to-peer network using a PageRank-like vector based on rankings given by nodes. See [3].

6.2 BarterCast (2009)

BarterCast uses MaxFlow calculations in two directions to judge a potential peer as either a free-rider or contributer. See [4].

Chapter 7

Partial Solutions

7.1 Identity and secrets

7.1.1 Public-key cryptography

We often refer to the "identity" of an agent or network participant. We use this term to refer to the keypair that a node generates, and by which it identifies itself. Such a keypair consists of a public and private part: the first publicly known, the latter secret. Those in possession of a private key can demonstrate its ownership by *signing* pieces of data. Others can then use the public part of the key to verify the correctness of the signature. The keypair can also be used for encryption: all who know a public key can use it to encrypt a piece of information in such a way that only the holder of the corresponding private key can decrypt it.

7.1.2 Atomic transactions

One of the many assumptions made in section 3.1 was the atomicity of transactions. This appears trivial at first, but is not in practice. How should one transfer value at the same time as one records this value transfer? A transaction participant could refuse to perform work after a transaction has been signed, or an actor receiving work could refuse to sign a transaction after the work has been completed.

A simple scheme

For this reason we provide an example of a simple transaction scheme, for which it is beneficial for participant, not to cheat. The protocol consists of several steps. Participant A receives and pays for work performed by B.

1. A provides cryptographic proof to B demonstrating it agrees to a price for a specific good
2. B performs work and *obtains a cryptographic proof of work performed*
3. B publishes proof of work completed
4. Now either
   (a) A and B sign a transaction and complete the procedure
   (b) A refuses to "pay" B by signing the transaction, and B proves to the network that A cheated

Naturally, being proven to be a cheater should negatively impact reputation. Note how B can't blame A without both the proof of A willing to receive work, and the proof of B having performed the work.

At step 2 we skipped over how to exactly provide cryptographic proof of work performed. This type of proof should depend on the application. It should however always be at least as hard to obtain the proof as to perform the actual work.

Proof of transmission

In the case of file transfers, it is impossible to cryptographically prove that data has been sent over a passive channel [5]. It is however possible for an agent to prove to another that it is in possession of the data to be sent. This is called a "proof of custody", which is essentially a zero-knowledge proof which can only be performed by an agent in possession of a private key.

7.2 Double-spend and fork prevention

An agent A might try to benefit by using its current chain to compel another agent B to make a (large) transaction. After the transaction has been made, A can hide the transaction made with B from C, with whom he then does a second (large) transaction which C would not have done if it had know about the first transaction. This is also known as "double spending".

In order to prevent an agent from making a transaction while withholding previous transactions, those agents collecting blocks should always look for two blocks made by the same agent pointing to the same block: this implies a double-spend. Once a double-spend is detected, and the author of the transaction is marked as a "cheater", agents should refuse to transact with this person again.

7.3 Sybil prevention

7.3.1 Fundamental Limitations

It can be shown that Sybil attacks cannot always be prevented. For instance, when using a reputation function that only depends on the topology of the network graph, it can be shown that such a system is always susceptible to Sybil attacks [6]. This means relative positions, for instance compared to some trusted user, need to be used. Even in this case, there are limitations to preventing Sybil attacks. Nevertheless, it might suffice to make Sybil attacks unprofitable, instead of completely eliminating the possibility of such an attack.

7.3.2 Proof-of-work system for identity creation

While generating a keypair is cheap, this can be made more costly by imposing certain requirements on the keypair. For example, a protocol could require all public keys to have a certain low hash value.

Depending on several factors, this measure can be effective. This measure is limited by the low cost with which honest users should be able to create a keypair. A reasonable requirement for keypair generation could be "within three seconds on a smartphone". In that case, a determined attacker can still easily generate an enormous number of identities with general-purpose computing on graphics processing units.

7.3.3 Tying identities to IP-addresses

As suggested in [4], it is possible to link identities to IP addresses, allowing only a fixed number of identities to exist for every IP address.

7.3.4 Latency

By requiring a limited link latency for all agents interacted with, participants can force the geographical proximity of their transaction partners. Otherwise they could ask another trusted agent to test their geographical proximity to the agent being considered for a transaction.

7.3.5 NetFlow

When sybils are being used by attackers to feign contribution to the network, there is typically much traffic between the attacker and its pseudonymous identities, and less across the network. Therefore, the NetFlow algorithm uses max-flow computations to determine the traffic flow across the whole network, serving as a metric for trustworthiness of agents in the network. Hence, the profit of cheating using Sybil effects is bounded. However, the NetFlow algorithm is computationally expensive, which means it is not fast enough for most practical applications. The algorithm could possibly be reconsidered, or replaced by a (probabilistic) approximation variant.

7.4 Consensus

7.4.1 Checo

Chapter 8

Further research

We identify unanswered questions several area's of future research. This chapter provides an overview of these questions and attempts to structure them. Note that barely any of the differences between the theory and practice as highlighted in chapter 5 are solved problems, let alone the interplay between them.

8.1 Identity and secrets

8.1.1 Derived identities

If one is willing to accept a third, central party as a trusted distributor of identities only, such as a government, it may be possible to address the sybil attack problem as follows. Use the identity (public-private keypair) as received from the trusted third party to create a derived identity for a specific application. We now need a zero-knowledge proof scheme to show that any derived identity is backed by a master identity, which can't be produced in large quantities. Moreover, a zero-knowledge proof is needed to demonstrate that two derived identities are not derived from the same master identity, such that for any specific application, only a single derived identity can be created by each master identity. In this way, users can be anonymous while be certain that no individual has multiple identities.

8.1.2 Proof of transmission

In section 7.1.2 it was mentioned that a proof of transmission over a *passive* channel is impossible. A passive channel is one in which "does not modify or sign the data it transmits" [5]. It may however be possible to realize such a proof using other nodes in the network.

8.2 Sybil prevention

8.2.1 Computational feasibility of NetFlow

While NetFlow gives strong guarantees for sybil resistance, the computational complexity of the algorithm is too great [2]. NetFlow could, for example,

- be approximated, while its sybil defense mechanism deteriorates only an acceptable amount, or

- be extended with the exchange of results of computations.

8.2.2 PageRank

While PageRank is relatively cheap to approximate precisely, it is unclear what guarantees it provides, if any. Among the open questions about PageRank are:

- What are the different metrics we can calculate the PageRank score on, and what do they tell us about the participants?
- Does the PageRank score provide any (probabilistic) guarantees?
- Should temporal aspects be included in this computation? How?
- How could different PageRank scores, computed on different network metrics, complement each other?

8.2.3 About time

An open question is how to prioritize older transactions made. Should they be removed from computations, made less important, or be regarded just as important as the most recent transactions? This is interesting from both a normative point of view, as well as a utilitarian point of view considering changes in effectiveness of for example sybil prevention.

8.3 Advanced concepts

8.3.1 Smart contracts

It's interesting to consider what an execution model for a DAG blockchain or tangle would look like. The triggering of logical rules using state change can't be synchronized globally, as this would require global consensus. Instead, alternative models need to be defined.

Appendix A

Materials to be integrated

The body of research on the topic of this book is of decent size and continuously growing. This appendix lists some works which have not yet been integrated in the rest of this document, but should be.

A.1 Historical perspective

A.2 Incentives and game theory

A.3 Distributed systems fundamentals

A.4 Systems of trust with imperfect properties

Bibliography

[1] Trust (emotion) - Wikipedia, The Free Encyclopedia.

[2] Pim Otte. Sybil-resistant trust mechanisms in distributed systems. Master's thesis, Delft University of Technology, December 2016.

[3] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The EigenTrust Algorithm for Reputation Management in P2P Networks. In *Proceedings of the 12th International Conference on World Wide Web*, WWW '03, pages 640–651, New York, NY, USA, 2003. ACM.

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

[5] Pavel Kravchenko and Vlad Zamfir. Cryptographic proof of custody for incentivized fileshare-ing.

[6] Alice Cheng and Eric Friedman. Sybilproof reputation mechanisms, 2005.

Index