TrustChain: A Sybil-resistant Scalable Blockchain

Pim Otte, Martijn de Vos, Johan Pouwelse

Delft University of Technology, Delft, The Netherlands

Abstract

TrustChain is capable of creating trusted transactions among strangers without central control. This enables new areas of blockchain use, such as providing a free alternative for the Uber and Airbnb matchmaking platforms. Our approach offers scalability, openness, Sybil-resistance, protection against the 51% attack and removes the need for proof-of-work. TrustChain is a permissionless tamper-proof data structure for storing transaction records of agents. It is inherently parallel and every agent creates his own genesis block. TrustChain includes a novel Sybil-resistant algorithm named *NetFlow* to calculate trustworthiness of agents. Our formal proof demonstrates that irrefutable historical transaction records offer security and seamless scalability, without requiring global consensus.

We show how the tragedy-of-the-commons can be addressed using TrustChain. We create an immutable chain of temporally ordered interactions for each agent with TrustChain. NetFlow ensures that agents who take resources from the community also contribute back. Based on real-world deployment we show by using our extracted data that TrustChain has sufficient informativeness to identify freeriders, leading to refusal of service. We present an operational decentralized market to buy and sell any asset, like community contributions. The evidence in this paper demonstrates the viability of TrustChain to enable a non-profit version of the sharing economy.

Keywords: blockchain, trust, reputation, tamper-proof data structure, tragedy-of-the-commons

2010 MSC: 00-01, 99-00

1. Introduction

We present TrustChain, technology specifically designed to create trust. In essence, trust is a feeling of security, based on the belief that someone or something is knowledgeable, reliable, good, honest, and effective. The blockchain is said to be a breakthrough in computer science that holds the promise of reducing the cost of establishing and maintaining trust for both individuals and organizations [1]. The blockchain lets people who have no particular confidence in each other collaborate without having to go through a neutral central authority.

Our TrustChain work contributes to an ambitious goal: establishing a *generic* method to create trust. A generic method can be re-used in a wide range of contexts, can be applied in varying fields of applications, and provides a shared common infrastructure. The main TrustChain design principle is having entities that would otherwise not trust each other agree on a common record of events in a continuous growing process. We use cryptographically signed records of events to create an unbroken, irrefutable, and sequential chain of evidence. We combine this chain-of-evidence with proven approaches from sharing economy companies to create trust. The key novelty of our work is an experimentally-validated blockchain, capable of utilising proven methods to create trust.

The Internet has altered our feeling of trust. The heated debate from a decade ago, centered around the question if unvetted volunteers could write a reliable encyclopedia, has cooled down. Uber invites people to use complete strangers for taxi services. Airbnb is based on the idea that you let unknown travellers from far away places sleep in your home, without worries. Trust is the essential problem to address in markets and in sustainable online communities in general. The breakthrough that made the *sharing economy* possible has been markets with trust. Reputation systems collect and process information about past interactions to help people evaluate the trustworthiness of others [2]. However, the architecture of the sharing economy leads to significant inefficiencies which make it unsuitable for broader usage. It does not provide a generic and future-proof method to create trust.

Ironically, companies in the sharing economy do not share reputations. A solid reputation on one platform is locked into that platform: you cannot move your reputation. Like other forms of lock-in, this inhibits competition and encourages the growth of monopolies. Additionally, when a platform goes out of business, the accumulated reputation and trust of the users are likely to be lost forever. Exporting your reputation is hardly ever possible. Sharing economy companies rarely disclose details of their security measures. The trust and security of these various platforms is thus almost impossible to share, re-use, or merge into a common infrastructure. Currently, each new service requiring trust needs to re-solve the same known issues. A common solution is likely to reduce platform lock-in effects and increase competition. The TrustChain concept is therefore disruptive to the disruptors that form the sharing economy. A shared infrastructure for creating trust is even likely to be more resilient against abuse. We envision a decentralized infrastructure with significant networking effects, such as the Internet itself, one in which no central hub acts as a gatekeeper of trust.

The problem is that to date, scientists have never managed to create a self-organising mechanism to create trust, resilient against all known types of attack (e.g. replay, man-in-the-middle, ballot-stuffing, slander, eclipse, and the Sybil attack). The most challenging attack is the Sybil attack, in which adversaries create numerous fake identities to gain a disproportionately large influence [3]. Traditional defences against Sybil attacks rely on validated identities issued by a trusted authority. Live reputation systems used by millions of people *without exception* rely on a single point of control (and platform lock-in), for instance, eBay auctions [4], Amazon reviews [5] and Google searches [6]. The requirement for agents to present a trusted identity conflicts with the need for permissionless open membership.

We present an approach which provides true distributed trust, void of any gatekeeper, while still providing strict bounds on the profitability of Sybil attacks. TrustChain is a remarkably simple blockchain-based data structure. It can be used to record transactions and to make these transactions tamper-proof. Our design specifically strives for simplicity and avoids complexity. TrustChain gives each agent in the network their own chain of transactions. Using what are essentially parallel chains yields inherent scalability, removes the need for a proof-of-work mechanism, global transaction broadcasts, leadership elections, forks, permissions or sharding. NetFlow is our algorithm to calculate the trustworthiness of agents with Sybil resistance using prior transactions as input.

To demonstrate the strength of TrustChain we build two applications on top of it. First, we created an online community of volunteers which share Internet bandwidth and address the tragedy-of-the-commons problem. Second, we crafted a decentralized market as an alternative for central matchmaking platforms such as Uber and Airbnb.

The contributions of this work are the following:

- A tamper-proof, scalable and blockchain-based data structure (TrustChain).
- A Sybil-resistant model to calculate trustworthiness (NetFlow).
- A tragedy-of-the-commons experiment which addresses freeriding.
- A decentralized and self-regulated market.

Our work includes both a formal proof of Sybil resistance and an Internet deployment of TrustChain, NetFlow, freeriding prevention, and a decentral market.

2. TrustChain Architecture

TrustChain facilitates trustworthy transactions, instead of the more traditional cybercurrency transfers. Our work differs significantly from related blockchain technology. Bitcoin shows an elegant solution to the double spending problem. As well as keeping track of who owns every bitcoin today, it also contains a record of who has owned every bitcoin since its inception. Unfortunately, the requirement of a global, consistent state does not scale and requires

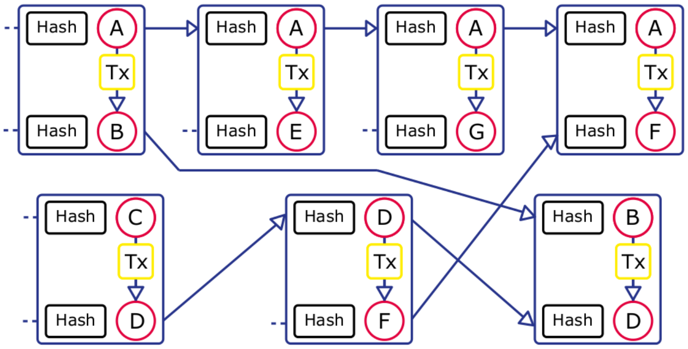


Figure 1: The tamper-proof TrustChain data structure to record transactions.

additions such as leaders or supervisory servers [7]. Numerous scientists contributed to the effort of creating a currency that scales [8, 9, 10]. In the past decade we have designed and deployed fully distributed accounting systems with seamless scalability [11, 12]. Our early deployment in August 2007 of a primitive ledger called BarterCast pre-dates Bitcoin [13]. Our experience enabled us to facilitate the creation of trust in a setting with full self-organisation.

The general architecture of TrustChain is illustrated in Figure 1. Each element in the data structure represents a transaction between two participants. When initiating new transactions, each participant includes the hash of their last interaction, creating a chain of temporally ordered interactions. In addition, each record is provided with a sequence number that is unique in the chain. TrustChain records are cryptographically signed by both parties and together form a directed graph.

This agreement by *both parties* is a key novelty of TrustChain and a fundamental difference between our approach and related blockchain work. The remarkable simplicity of TrustChain is a direct consequence of the hard requirement for agreement by both participating parties. TrustChain records are tamper-proof and irrefutable since modification of a specific record can be detected by verification of the cryptographic signature. This removes the need for Bitcoin-like global consensus mechanisms and competing forks which may invalidate transactions. The vulnerability of existing blockchains to the 51 %

attack is also addressed. The validity of a block of transactions is easy to establish and immediate, confirmation time is avoided. We call this a *bottom-up consensus model*.

In a directed graph representation each historical encounter in TrustChain has two incoming edges and two outgoing edges. Violations of these rules can be efficiently detected, for instance, signing two records with the same prior record pointer (see Section 5). TrustChain creates what we define as *entanglement*: each participant quickly becomes intertwined ("entangled") with others as more records are created. This is another property which makes detecting fraud easier.

TrustChain records are designed to be exchanged by agents using gossiping and replicated widely. Each agent operates their own bulk storage of records, resulting in the partial storage of the global directed graph. Every agent publishes their own unique chain. Their first recorded transaction is their genesis block.

Each agent in our system monitors the interactions by others and collects TrustChain records to compute trustworthiness levels. Collecting this information is challenging for the agents due to their vulnerability to various attacks, their limited resources, and the burst of their interactions. Our prior work investigates an attack-resilient and scalable solution to this collection problem using random walks and similarity functions [14, 15].

In this work we assume that this specific collection problem is addressed: we focus on the challenging problem of Sybil-resilient calculation of trust scores and operational systems built upon TrustChain.

3. Formal Model

We now present the NetFlow mechanism and proof its resistance to Sybil attacks. In order to do so, we start with a more abstract mathematical representation of the TrustChain: the ordered interaction model. Then we built upon this model to define the Sybil attack, which then can be used to show that the NetFlow mechanism is indeed Sybil-resistant.

3.1. Interaction Models

We consider a distributed work system that consists of n agents, each capable of doing work for each other. Before information about interactions can be exploited, we first define an interaction model which includes temporal information about interactions between agents in our network [16].

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

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

Note that $w(i, p) = 0$ *must hold if* $p \notin a(i)$*.*

Furthermore, for each $p \in P$*, the following set is completely ordered.*

$$I_p = \{i \in I : p \in a(i)\} \tag{1}$$

The ordering on I_p *is denoted by* $\leq_p$*.*

Agents represent entities that can interact with each other. An interaction involves two different agents, one or both performing work for each other. Definition 1 can be applied to almost every network that considers interactions and a quantitative amount of work performed between agents. The only other necessity is that something induces an order on the interactions of each agent. In the case of TrustChain, this is induced by the time at which an interaction is signed and stored.

Definition 2 (Successor of an interaction). *Let* $M = \langle P, I, a, w \rangle$ *be an ordered interaction model. Let* $p \in P$*,* $i, j \in I$ *such that* $i <_p j$*. If there exists no* i' *such that* $i <_p i' <_p j$*, then* j *is the successor of* i*. This is denoted by* $i \preceq_p j$*.*

The ordered interaction model can be applied directly to the TrustChain data structure presented in Section 2. Each record in the data structure is represented by an interaction between two agents. In the TrustChain data structure, the ordering on I_p should be determined by the order of records in time. However since this cannot be enforced by the system, each participant also includes the hash of their previous interaction, creating a chain of temporally ordered and tamper-resistant interactions for each participant. This order is the definitive order of I_p for each agent p. If agents sign and store their interaction at the same time, the orders on the sets I_p extend in a natural way to an order on I.

Definition 3 (Interaction graph). *Let $M = \langle P, I, a, w \rangle$ be an ordered interaction model. The weighted interaction graph G_M is defined as follows:*

- $V := \{v_p : p \in P\}$

- $E := \{(v_p, v_q) : \exists i \in I, a(i) = (p, q)\}$

Let $(v_p, v_q) \in A$, then the weight w of (v_p, v_q) is equal to

$$w((v_p, v_q)) := \sum_{i \in I : a(i) = (p,q)} w(i, p) \tag{2}$$

The interaction graph represents the total contributions of agents to each other and is commonly found in literature when considering networks where agents are collaborating to achieve some common goal.

The following definitions are inspired by the work of Seuken and Parkes [17].

Definition 4 (Subjective Work Graph). *A subjective work graph from agent i's perspective, $G_i = (V_i, E_i, w_i)$, is an interaction graph derived from an ordered interaction model M_i, where the set of interactions in the model is a subset of all interactions that includes at least all interactions of i.*

The subjective work graph allows us to model the situation in which agents have a partial view on the network. Note that this cannot be considered as a subgraph of the interaction graph derived from the complete model, as the weights on the edges might differ.

Definition 5 (Choice Set). *The choice set $C_i \subseteq V_i \setminus \{i\}$ for agent i is the set of agents that are currently interested in receiving some work from i.*

We can compute a score for each agent in the choice set with an accounting mechanism.

Definition 6 (Accounting Mechanism). *An accounting mechanism M takes as input a work subjective graph G_i and determines the score $S_j^M(G_i, C_i) \in \mathbb{R}$, for any agent $j \in V_i$, as viewed by agent i.*

Definition 6 defines what is basically a scoring mechanism. This can then be used to select an agent from the choice set to deliver work to.

Definition 7 (Allocation Policy). *Given G_i, choice set C_i and an accounting mechanism M, an allocation policy A is a function that maps these three objects to an agent $j \in C_i$. This choice is denoted by $A(S^M(G_i, C_i))$. This agent is chosen to receive a unit of work from i.*

Accounting mechanisms have various properties which will now be defined and discussed.

Property 1 (Independence of Disconnected Agents (IDA)). *Let M be an accounting mechanism, let $G_i = (V_i, E_i, w_i)$ be a subjective work graph and let C_i be a choice set. Agent $k \in V_i$ is a disconnected agent, if for this agent there are no edges in E_i, or for this agent all edges in E_i have zero weight. By $G_i' = (V_i', E_i', w_i')$ we denote the subjective work graph with a disconnected agent k removed.*

M satisfies "Independence of Disconnected Agents" if for any disconnected agent k:

$$\forall j \in V_i' : S_{ij}^M(G_i, C_i) = S_{ij}^M(G_i', C_i') \tag{3}$$

Property 1 basically states that adding or removing agents with no work consumed or performed (i.e. without connections to the rest of the network) does not influence the scores of other agents in the network.

Property 2 (Anonymity (ANON)). *Let M be an accounting mechanism, let $G_i = (V_i, E_i, w_i)$ be a subjective work graph, let C_i be a choice set and let f be a graph isomorphism such that $G'_i = f(G_i)$, $C'_i = f(C_i)$. M satisfies anonymity if for every such isomorphism:*

$$\forall j \in V_i \setminus \{i\} : S^M_{ij}(G_i, C_i) = S^M_{f(i)f(j)}(G'_i, C'_i) \tag{4}$$

Essentially, property 2 states that scores can only be based on the structure of the subjective work graph. We can imagine situations where this property does not hold, when additional information about an agent apart from the work performed and consumed is used to determine scores. However, this is not applicable to the scenarios that we consider in this research.

Property 3 (Weak Transitive Trust). *Accounting Mechanism M satisfies weak transitive trust if, for every work graph $G_i = (V_i, E_i, w_i)$, there exists a $j \in V_i$, after adding node k to $G_i = (V_i, E_i, w_i)$, this leads to $G'_i = (V'_i, E_i, w_i)$ with $V'_i = V_i \cup \{k\}$, and there exists an amount of work W_j and W_k such that if j performs W_j units of work for i, and k performs W_k units of work for j, a report of which leads to a work graph G''_i representing this, then for every choice set C_k that contains k, but not j, it holds that $A(S^M(G''_i, C_k)) = k$.*

This property basically implies the transitivity of trust. In other words, if an agent A is helped by agent B and agent B by agent C, then A will value the contributions of agent C.

A successful Sybil attack by a set of sybils increases their own reputation or lowers the reputation of other agents by initiating interactions with agents in the network.

Definition 8 (Sybil attack). *Given a subjective work graph $G_i = (V_i, E_i, w_i)$. A Sybil attack by agents $J \subseteq V_i$ is a tuple $\sigma_J = (V_s, E_s, w_s)$ where $V_s = \{s_{J_1}, s_{J_2}, \ldots\}$ is a set of Sybils, $E_s = \{(x, y) : x, y \in V_s \cup J\}$, and w_s are the edge weights for the edges in E_s. Applying the Sybil attack to agent i's subjective work graph $G_i = (V_i, E_i, w_i)$ results in a modified graph $G_i \downarrow \sigma_J = G_i' = (V_i \cup V_s, E_i \cup E_s, w')$, where $w'(e) = w_i(e)$ for $e \in E_i$ and $w'(e) = w_s(e)$ for $e \in E_s$.*

Definition 9 (Sybil attack profit). *Let G_i be a subjective work graph. For all $j \in \mathbb{N}$, let $(\sigma_J)_j$ be a Sybil attack on $(G_i)_j$, where $(G_i)_0 := G_i$ and $(G_i)_j$ for $j > 0$ is defined by the subjective work graph that consists of $(G_i)_{j-1} \downarrow (\sigma_J)_j$ and the assignment of one unit of work to $A(S^M((G_i)_{j-1} \downarrow (\sigma_J)_j, C_i))$.*

Let ω_-^n be the sum of the amount of the work agents in J have performed for the network after n steps, including work performed before the start of the Sybil attack. Let ω_+^n be the amount of work that agents in J or any of their Sybils obtain from the network. Any work obtained before the start of the Sybil attack is disregarded.

The profit of this sequence of Sybil attacks is:

$$\sup\{\frac{\omega_+^n}{\omega_-^n} : n \in \mathbb{N}, \omega_-^n \neq 0\} \tag{5}$$

If this supremum is infinite, the Sybil attack is strongly beneficial.

If the supremum is finite and is larger than 1, the Sybil attack is profitably weakly beneficial.

If the supremum exists and is smaller than or equal to 1, the Sybil attack is unprofitably weakly beneficial. This case is also known as "contributing to the network".

3.2. NetFlow: Bounding Sybil Attack Profit

We now present an accounting mechanism that is partially resistant to Sybil attacks, in the sense that it can be profitably weakly beneficial, with bounded profit.

Definition 10 (NetFlow limited contribution). *The NetFlow limited contribution accounting mechanism M is defined as follows: given a subjective work graph $G_i = (V_i, E_i, w_i)$ and choice set C_i, agent i computes the following for each agent j: $c_j := \max\{MF_{G_i}(j,i) - MF_{G_i}(i,j), 0\}$, where $MF_{G_i}(i,j)$ denotes the value of the maximum flow from i to j in G_i, where the weights are the capacities on the arcs.*

Let G_i^N be the graph G_i modified with c_j as node capacities for each node, except for c_i which should be infinite. We now assign a score to agent j as follows: $S_j^M(G_i, C_i) = MF_{G_i^N}(j,i)$.

Theorem 1 (Properties and Sybil-resistance of NetFlow). *NetFlow as an accounting mechanism with a complete work graph satisfies the IDA, ANON and weak transitive trust properties and is robust against profitably weakly beneficial Sybil attacks.*

Proof 1. *IDA follows from the fact that flow to and from agents does not change if independent agents are added or removed. ANON follows from the observation that relabelling nodes will not change the flows and thus the assigned scores of agents in the graph.*

We now consider the weak transitive trust property. Let k be the node added and let $M := \max_{j \in C_i}\{S_j^M(G_i, C_i)\}$ be the largest amount of reputation in the system. Then using $W_j = M + 1$ and $W_k = M + 1$ satisfies the requirements of weak transitive trust.

This leaves resistance against profitably weakly beneficial Sybil attacks. Consider the amount of work performed by the agents in J after n steps of the Sybil attack, ω_-^n. For each unit of work that is contributed to an agent in J during the Sybil attack, the capacity of some agent in J that has directly contributed to the network must drop by 1 unit, possibly more agents in J. This means that at most ω_-^n units of work can be contributed to agents in J after n steps. Therefore, $\omega_+^n \leq \omega_-^n$, and thus:

$$\sup\{\frac{\omega_+^n}{\omega_-^n} \ : n \in \mathbb{N}, \omega_-^n \neq 0\} \leq 1 \tag{6}$$

Since the computing agent possesses the complete work graph, no work can leak outside of this graph.

In the NetFlow mechanism, only agents that have a strictly positive contribution in terms of flow will induce network effects. In existing networks, this subset of the population tends to be fairly small. If most agents are assigned a zero score, the mechanism does not provide any information about them. This leads to the definition of informativeness.

Definition 11 (Informativeness). *Given an accounting mechanism M and a subjective work graph G_i, the informativeness of M given G_i is the fraction of agents in G_i that have a non-zero score.*

An idea to improve the informativeness would be to weight the work performed by the other agents higher than work consumed by them. The rationale behind this scheme is that agents that perform less work than they consume are tolerated to some degree. This is a trade-off in the sense that this will allow weakly profitable Sybil attacks in exchange for increased informativeness of the mechanism. In particular, a scaling mechanism would need to guarantee that it is impossible to leverage the scaling behaviour to a strongly profitable Sybil attack.

We scale NetFlow by rating work performed by the calculating agent i lower than other work, which is equivalent to rating all other work higher.

Definition 12 (α-NetFlow limited contribution). *Given a subjective work graph $G_i = (V_i, E_i, w_i)$, a choice set C_i, $\alpha \geq 1$, the α-scaled weights $w_{i,\alpha}$ are defined as follows:*

$$w_{i,\alpha}(e) = \begin{cases} \frac{w_i(e)}{\alpha} & \textit{if } e = (i,j) \textit{ with } j \in V_i \\ w_i(e) & \textit{otherwise} \end{cases} \tag{7}$$

The α-NetFlow accounting mechanism is computed as the NetFlow accounting mechanism, but on the graph $G'_i = (V_i, E_i, w_{i,\alpha})$.

As a shorthand, this mechanism is denoted by α-NetFlow.

It is trivial to verify that α-NetFlow also satisfies IDA and ANON. Furthermore, it has a similar, but slightly weaker defence against Sybil attacks.

Theorem 2 (Sybil-resistance of α-NetFlow). *α-NetFlow is resistant against weakly beneficial Sybil attacks with profit more than α.*

Proof 2. *Since NetFlow is resistant against weakly beneficial Sybil attacks with profit at most 1 (see Theorem 1) and the only difference between the two systems is that the contributions of agent i are decreased by a factor α, it must be the case that the profit of a Sybil attack can be at most α. After all, all altered interactions are with i and are therefore known to have actually happened. Concluding, the resource consumption of any party can be at most a factor α more than allowed by net flow, which implies an upper bound on the Sybil attack profit.*

4. Theoretical Performance of NetFlow

We now consider the theoretical performance of the NetFlow mechanism discussed in Section 3. In order to compute the score of one other node with NetFlow, up to $2n + 1$ max-flow computations are necessary, where n is the number of nodes in a subjective work graph G_i. In case one would compute the scores of all other nodes, $3n$ max-flow computations are needed. Hence, the performance of NetFlow will depend on the max-flow algorithm used and then incur another factor n of computational complexity. This could be mitigated by finding a way to compute multiple flows at the same time. In this section, possibilities to improve the performance of NetFlow will be explored. n denotes the number of nodes and m denotes the number of edges in the graph.

Well known algorithms for max-flow calculation in a network include Ford-Fulkerson, with computational complexity $O(m|f|)$, where $|f|$ is the magnitude of the maximum flow [18]. This is a pseudo-polynomial algorithm, since the complexity depends on the magnitude of numbers in the instance. The Edmonds-Karp algorithm specifies the order in which augmenting paths are considered, namely by doing a breadth-first search [19]. This allows the complexity to be pinned to $O(nm^2)$.

Dinitz' algorithm, (also known as Dinic's algorithm) functions in $O(n^2m)$ or $O(nm\log(n))$ if implemented with dynamic trees[20]. Goldberg and Tarjan introduced the preflow-push algorithm [21] which has a worst-case complexity of $O(n^2m)$. Again, this can be reduced to $O(nm\log\frac{n^2}{m})$ by using dynamic trees.

Work by King, Rao and Tarjan has resulted in an algorithm of order $O(nm\log_{\frac{m}{n\log(n)}} n)$ [22]. In combination with work by Orlin, this results in an $O(nm)$ algorithm for max-flow [23]. This is the current state-of-the-art when considering single source-sink max-flow computations.

When we consider all-pairs max-flow, one might think a speed-up is possible. Building a Gomory-Hu tree yields a method for which $n-1$ max-flow computations suffice [24], however, this method only works for undirected graphs. According to Ahuja, Magnati and Orlin, no method is known for all-pairs max-flow on directed graphs that uses less than $O(n^2)$ max-flow computations [25]. This work is from 1993, and to our knowledge this has not changed since. Note that for the application of NetFlow, it would be necessary to compute all flows with one fixed sink or one fixed source, which is not quite the same as the all-pairs max-flow problem.

The above research implies that using state of the art algorithms, a NetFlow algorithm implemented with current state-of-the-art algorithms would be at least $O(n^2m)$ in the worst case. Our implementation uses the preflow-push algorithm, yielding a worst-case complexity of $O(n^3m)$.

5. Security Analysis

We will now describe two attacks on the TrustChain where a malicious agent attempts to gain an unfair advantage by tampering with the data structure described in Section 2.

Branching Attack — Similar to the double spending attack in Bitcoin, agents in the network can create a branch of their own chain by creating two records with the same sequence number. This attack is illustrated in Figure 2 where the TrustChain structure of three agents is presented. Malicious agent M is

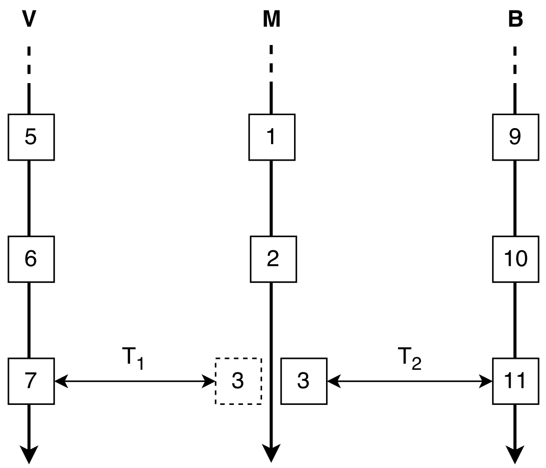


Figure 2: An example of a *branching attack* performed by agent M. The number inside each record indicates the sequence number. Agent M wishes to hide the dashed record.

involved in two transactions with agents V and B and creates records with the same sequence number for both transactions. Suppose that M intends to hide one of these two created records and exclusively spreads knowledge about the other record in the network. An intention to do so might occur when an agent performs a significant amount of work in transaction T_2 and consumes work in transaction T_1. Hiding the transaction that indicates consumption of work will have a positive impact on the reputation of agent M.

While this attack might seem successful at a first glance, the fact that agent M is hiding one of his transactions will eventually be detected when another agent, say S, learns about the historical encounters of agent V. During the verification of the chain of V, the hidden transaction of M is discovered which contradicts the knowledge that S has about transactions in which M has been involved. The two records that agent M has created together form a *proof-of-lying* and should be broadcast in the network. Other agents can verify the fraudulent actions of M with little computational effort.

Replay Attack — During a replay attack, a malicious agent reuses the pointer to a record of the other party. This process is presented in Figure 3. The

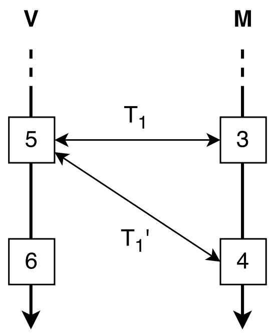


Figure 3: An example of a *replay attack* performed by agent M. The number inside each block indicates the sequence number.

motivation behind this attack is that a malicious agent M can claim that he has performed a transaction that is beneficial for him twice. This attack is relatively easy to discover: when verifying the correctness of the chain of agent M, we can detect whether there are two records that have an outgoing edge to the same record of the other party involved in the transaction. Note that there can be an arbitrary number of (valid) records between the two records with respectively sequence number 3 and 4 in the interaction history of agent M. In this attack, the proof-of-lying consists of the two blocks created by M; any agent in the network can verify the invalidity by observing the outgoing edges of the records at issue.

6. Tragedy-of-the-commons Experimentations

One of the fundamental questions of our time is dealing with the needs of human beings and preserving our finite natural resources. Online communities are similar to our natural world in a sense that they require sustainable management of resources such as storage, Internet bandwidth, and computational power. We now apply TrustChain to address *tragedy-of-the-commons* issues without the need for any central authority [26]. The tragedy is that, in the absence of regulation, each individual will have a tendency to exploit the commons to his/her own advantage, typically without limit. Under this state of affairs,

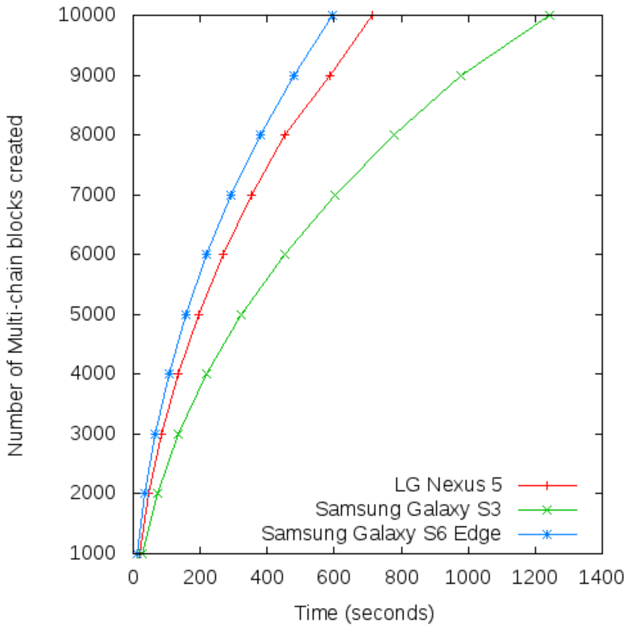


Figure 4: TrustChain record creation performance on mobile devices.

the commons is depleted and eventually ruined. Our experiments combine accounting of community contributions by agents, estimating the trustworthiness of strangers, identification of freeriders, and resistance against Sybil attacks, in the sense that they can be profitably weakly beneficial, with bounded profit.

6.1. TrustChain Overhead and Scalability

The ability to record community contributions by agents in an efficient, lightweight, and scalable manner is key. Recall that TrustChain records are tamperproof irrefutable records, cryptographically signed by both participating agents as described in Section 2. We now aim to quantify the overhead and scalability of TrustChain within a challenging environment with constrained computational resources: mobile devices.

We fully implemented TrustChain in the Python programming language and created a simple smartphone application that creates TrustChain records, all released as open source[1] [27]. Figure 4 shows the required time to create and store 10,000 TrustChain records on various types of mobile devices. The horizontal axis denotes the time into the experiment in seconds while the vertical axis specifies the number of TrustChain records that have been created. This experiment uses three Android-based devices to demonstrate the usability of TrustChain on low-cost and aged hardware, namely a Samsung Galaxy S3 (2012), a Nexus 5 (2013), and a Samsung Galaxy S6 (2015). At the start of this experiment the durable storage is empty and new records are generated and inserted quickly in the embedded database. With subsequent database growth the insertion cost somewhat increase. These results show that on ageing hardware it is possible to create thousands of records easily. Other experiments confirm that creating and retrieving records for hours continuously is also affordable.

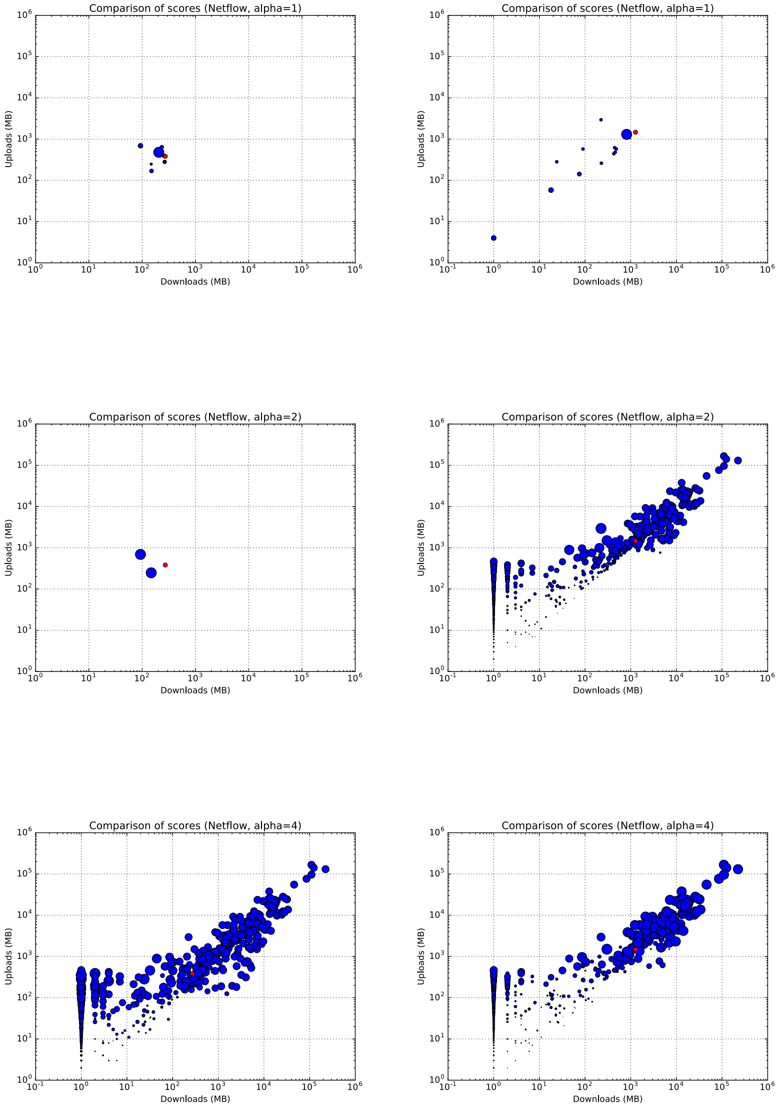
Note that *all* current Bitcoin transactions can be handled by a single low-end smartphone if creation of trust and global consensus did not require an extremely wasteful mechanism. Bitcoin is currently limited to approximately 7 transactions per second globally. Our performance levels indicate that TrustChain achieves record creation speeds of 16,000 transactions per second on modern hardware.

6.2. NetFlow Trustworthiness Estimation and Informativeness

Next, we focus on the evaluation of the Sybil-resistant NetFlow mechanism described in Section 3. We have performed a public experiment in which 917 volunteers recruited from the Internet participated in our open one-month tragedy-of-the-commons study. These volunteers installed and used our YouTube-like video-on-demand platform, called Tribler [28]. This software is part of our long running "bandwidth-as-a-currency" research line, with a first operational system deployed in August 2007 [13]. In prior experimental work we collaborated with Wikipedia.org and enabled bandwidth donations to their website [29].

Each of our volunteers operated a TrustChain implementation, created his

[1] https://github.com/Tribler/tribler/tree/e744e2ca/Tribler/community/multichain



(a) Computing agent contributed and consumed around 120MB, for $\alpha \in \{1, 2, 4\}$

(b) Computing agent contributed and consumed around 1050MB, for $\alpha \in \{1, 2, 4\}$

Figure 5: The NetFlow trustworthiness scores from the perspective of several agents. The sizes of bubbles indicate the assigned scores (different scales across plots). The red bubble specifies the computing agent.

own genesis block upon installing our software, and automatically advertised this publicly on the network. The 917 genesis blocks we discovered may not necessarily belong to unique individuals, since users may purposefully delete their identity or users may have installed the software on multiple machines. Within our Tribler video streaming applications users pool their bandwidth together and share it with others in a peer-to-peer fashion. When a volunteer gives bandwidth or takes bandwidth from any other volunteer within our study, it is cryptographically signed and recorded on TrustChain. We extracted a TrustChain dataset from Tribler. In Figure 5, we indicate each volunteer as a data-point. The horizontal axis denotes their bandwidth consumption from other agents in the community, and the vertical axis indicates their total bandwidth donation to others. The unit of both axes is megabyte (MB).

For each agent we calculate NetFlow-based trustworthiness scores using the mechanism given in Definition 10 (see Section 3). We first consider the influence of the scaling parameter α and how informative the mechanism explained in Section 3 is. Figure 5 presents the NetFlow scoring mechanism for two agents with different amounts of contributions, for three distinct values of α ($\alpha = 1$, $\alpha = 2$ and $\alpha = 4$). These agents have been selected by sorting the list of agents by their total amount of uploaded data, and taking the agents at the 70th and 80th percentile. The agent whose point of view is taken for the computation is marked red. Each bubble is an agent whose score is computed using the NetFlow mechanism. Note that the scale of the bubbles varies between the different plots. This is due to the fact that absolute sizing would result in indiscernible bubbles for computations from the perspective of agents with lower absolute upload and download amounts.

Upon inspection of these figures, several interesting observations can be identified. The first is that increasing the scaling factor α does indeed increase the informativeness of the mechanism, resulting in non-zero scores for more agents. In particular, note that for $\alpha = 1$ many of the agents with higher upload and download values have a zero score. This is likely due to the fact that scores of agents are not high if their contributions are limited by the consumption and

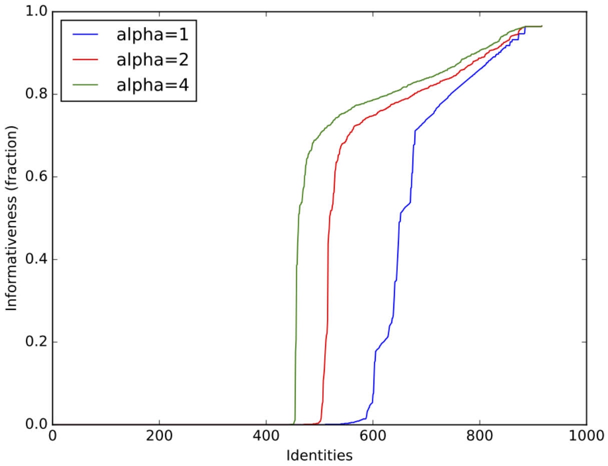


Figure 6: Informativeness for different values of α.

contribution of the agent that calculates the NetFlow scores. This effect decreases significantly: for $\alpha = 2$ and $\alpha = 4$, agents with absolute higher amounts of upload and download will often be assigned a positive score.

Increasing the value of α does come at a cost. It can be observed that for $\alpha = 1$ no agents that have a ratio of upload/download significantly below 1, have a positive score. On the contrary, for higher α, agents that contribute less than the calculating agent might still have a non-zero score if they have contributed about the same or more in absolute terms.

Let us now consider the informativeness of the NetFlow mechanism. If too many users are assigned a zero score, the mechanism does not actually yield a ranking.

Figure 6 provides informativeness curves with different values for α. This is constructed as follows: for each agent, the fraction of agents that have a positive score is computed. The lines in the plot are these fractions ordered from low to high. One part of the population will never be able to increase the informativeness by scaling. Hence, Figure 7 shows the same data, excluding agents that have not downloaded any data. Observe that as α increases, so

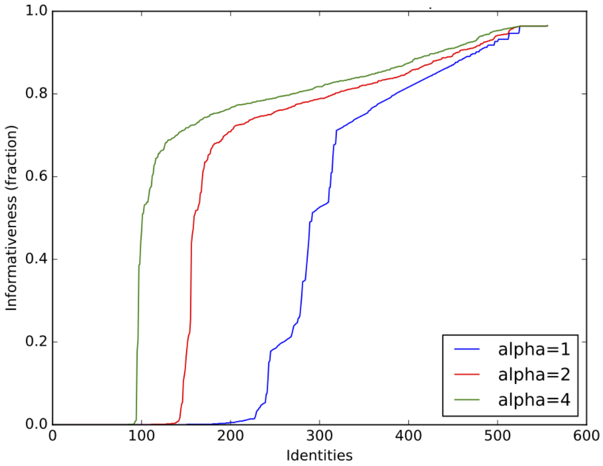


Figure 7: Informativeness for different values of α, excluding agents without downloaded data.

does the informativeness. Furthermore, for higher α the set of agents with 0 informativeness decreases in size. In addition, note that there is quite a sharp jump from 0 informativeness to around 0.7.

7. Decentralized Market

We now present a proof-of-concept of a decentralized, self-regulated market, capable of replacing existing centralized platforms like Airbnb and Uber. In order for our work to be broadly usable we focus on generic stock trading of any electronic asset. We designed, implemented, and tested a fully decentralized stock exchange, capable of operating an exchange market for cybercurrency. We test our proof-of-concept using a NASDAQ dataset and describe measures for theft protecting.

Our design is based on disseminating orders to buy and sell electronic assets. Each agent within this market gathers these orders and observes the state of the market. This approach requires no central server, oversight, or regulator.

Creation and dissemination of orders, which are present in the form of asks and bids, are key features. An ask order is created when a user wishes to sell assets whereas a bid is created when a user is interested in buying assets. When a new order is made, the corresponding bid or ask messages are gossiped through the network.

Each user keeps track of known outstanding orders in a limit order book that supports multiple price levels. An order matching engine has been integrated, allowing automated matching of orders in the limit order book without relying on any user intervention. The matching engine is executed when a new order is created by a user or when the client learns about orders created by other users.

When two users agree to trade with each other, the settlement phase proceeds in an incremental manner to decrease the effectiveness of theft in the case one of the transaction participants is malicious. This is achieved by splitting the transaction into n equal parts. We should note that while this scheme is feasible for assets that can be divided like currency and community contributions, it is less effective in the scenario of buying physical goods.

Our proof-of-concept implementation allows to buy and sell community contributions for bitcoin but is extendible to support trading more generic assets. Our implementation is built upon Dispersy, a platform to design communities which can be considered as an overlay network where agents discover each other and communicate [30]. To demonstrate the feasibility of our decentralized market design, we modified the implementation to support stock trading.

```
@0:0 start_dispersy
@0:1 online
@0:45.004241 bid 58533 18 {2}
@0:45.004261 ask 58532 24 {3}
@0:45.004447 bid 58531 14 {4}
```

Listing 1: A part of the scenario file used in the market experiment. Each line indicates an order creation, including time, type (bid/ask), price, quantity and the peer that executes this action.

For this experiment, the LOBSTER limit order book data set is used which

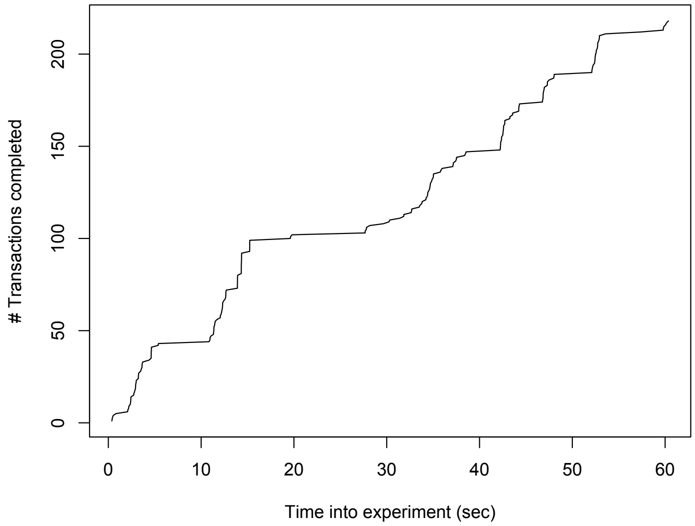


Figure 8: The number of completed transactions during the decentralized market experiment.

provide a list of events that modify the order book and includes time stamps of these events[31]. In particular, we use an order book evolution of the Apple stock (AAPL) since that symbol is traded in high volume. By dividing this data set over the market instances in our experiment using a round-robin allocation, we aim to mimic behaviour of the NASDAQ stock exchange, but without central authority. For the experiment, a scenario file is used that indicates the actions that instances have to execute at specified timestamps. Listing 1 shows a part of this scenario file. The DAS5 supercomputer is used to execute this experiment where we allocate ten nodes and run ten instances of the decentralized market on each node. During the scenario execution, 592 orders are created in total of which 336 ask orders and 256 bid orders. The creation of orders is started after a small delay to give the instances some time to bootstrap and discover other users in the network, yielding a higher market efficiency and a better message dissemination.

Figure 8 shows the number of completed transactions during the experiment. The horizontal axis denotes the time into the experiment in seconds whereas the vertical axis specifies the number of completed transactions. After the experiment is finished, a total of 218 transactions have been performed. This experiment shows the practical potential of a self-regulated, decentralized marketplace.

8. Related Work

The R3 Corda ledger work is closely related to TrustChain: it is also focussed on tamper-proof transaction recording. R3 Corda is one of the largest groups working on ledger technology deployment. Their key feature is "recording and managing the evolution of financial agreements and other shared data between two or more identifiable parties" [32]. Similar to our approach, they also avoid global consensus, proof-of-work, and fork mechanisms. Due to the R3 Corda focus on financial firms, a central element of their work is legally binding contracts between two parties and regulatory compliance. A key difference is that R3 Corda lacks gossiping and has no replication of records since transaction records are only stored by the two or more directly involved parties.

Both Bitcoin and Ethereum have been suffering from spam attacks in which attackers trigger a reduction in the rate of transaction block creation [33]. Flooding the overlay network is a key attack for prior proof-of-work based approaches. Our bottom-up based consensus model is vulnerable to flooding by numerous identities. From 2006—2012, there was much excitement in the research community about using social networks to mitigate Sybil attacks [34]. Algorithms such as SumUp, SybilGuard, SybilLimit, and SybilInfer provide resistance against sybils by analysing the social graph [35, 36, 37, 38, 39]. In 2007 we designed and Internet-deployed the first *fully* distributed reputation system that prevents lazy freeriding, called BarterCast [11, 12]. The BarterCast mechanism calculates reputation of agents by utilising a max-flow algorithm based on an agent's private history of its data exchanges as well as indirect information received from other agents. This work has been extended by Seuken and Parkes where the Drop-Edge accounting mechanism is introduced [40]. The same authors have added the notion of Sybil-proofness and transitive trust to Drop-Edge in subsequent research work [17].

There have been various proposals to create a more scalable blockchain. Bitcoin-NG is a blockchain protocol that is designed to scale and is Byzantine fault tolerant, robust to extreme churn and shares the same trust model obviating qualitative changes to the ecosystem [8]. Improving the block creation rate has been addressed by the GHOST rule, a modification to the way Bitcoin nodes construct and re-organize the blockchain [10]. While these models provide a significant increase in potential speed, they do not allow for unbounded scalability and require complex consensus mechanisms.

9. Conclusions

We demonstrated the viability for a new direction in blockchain research, a generic method to create trust. We enable new areas of blockchain usage, centered around the notion of trusted transactions. Our TrustChain work uses tamper-proof, temporal ordered, cryptographically signed transaction records to create irrefutable proof of past actions. By using a data structure that is resilient against various kinds of malicious behaviour, we illustrated it is possible to accurately record community contributions by agents. This enables mechanisms of self-reinforcing trust, by preferring agents who have been helpful to others in the past. The key novelty of TrustChain that it is remarkably simple, it is inspired by tit-for-tat.

Our key defence against attacks is NetFlow. The NetFlow mechanism yields a Sybil-resistant model to calculate the trustworthiness of agents and guarantee that agents who take resources from the community also contribute back. A formal proof is provided on the Sybil-resistance of NetFlow.

Our experimental results indicate that TrustChain is capable of addressing tragedy-of-the-commons problems without any central authority. With a one-month experiment involving 917 volunteers we have shown the both practical applicability and level of maturity of this work. We demonstrated the effectiveness of the NetFlow algorithm and proven that the informativeness is high enough to identify freeriders.

TrustChain in general is designed to be a scalable solution, in the current Python-based unoptimized form capable of verifying around 16,000 transactions per second using modern Android hardware. It avoids the double spending problem, does not require proof-of-work mechanisms, global transaction broadcasts, leadership elections, forks, permissions, sharding and central authority.

The breakthrough that made the sharing economy possible have been markets with trust. Our proof-of-concept implementation of a decentralized, self-regulated market demonstrates the feasibility of building a non-profit platform that is capable of replacing existing matchmaking companies like Airbnb and Uber. We envision a rich area of future research around trustworthiness using tamper-proof data structures. Future work will be focussed on a full-scale deployment of TrustChain in Tribler, maturing our work into an operational "bandwidth-as-a-currency" market inside Tribler, and formal verification of our bottom-up consensus model under development.

References

[1] T. Economist, The promise of the blockchain: The trust machine, `http://www.economist.com/news/leaders/21677198-technology-behind-bitcoin-could-transform-how-economy-works-trust-machine` (2015 (accessed December 30, 2016)).

[2] P. Resnick, R. Zeckhauser, Trust among strangers in internet transactions: Empirical analysis of ebays reputation system, The Economics of the Internet and E-commerce 11 (2) (2002) 23–25.

[3] J. R. Douceur, The sybil attack, in: 1st International Workshop on Peer-To-Peer Systems (IPTPS), Springer, 2002.

[4] P. Resnick, R. Zeckhauser, J. Swanson, K. Lockwood, The value of reputation on eBay: A controlled experiment, Experimental economics 9 (2) (2006) 79–101.

[5] S. M. Mudambi, D. Schuff, What makes a helpful review? a study of customer reviews on amazon. com, MIS quarterly 34 (1) (2010) 185–200.

[6] L. Page, S. Brin, R. Motwani, T. Winograd, The pagerank citation ranking: bringing order to the web.

[7] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. Gün, On scaling decentralized blockchains, in: Proc. 3rd Workshop on Bitcoin and Blockchain Research, 2016.

[8] I. Eyal, A. E. Gencer, E. G. Sirer, R. Van Renesse, Bitcoin-ng: A scalable blockchain protocol, in: 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16), 2016, pp. 45–59.

[9] Y. Lewenberg, Y. Sompolinsky, A. Zohar, Inclusive block chain protocols, in: International Conference on Financial Cryptography and Data Security, Springer, 2015, pp. 528–547.

[10] Y. Sompolinsky, A. Zohar, Secure high-rate transaction processing in bitcoin, in: International Conference on Financial Cryptography and Data Security, Springer, 2015, pp. 507–527.

[11] M. Meulpolder, J. Pouwelse, D. Epema, H. Sips, Bartercast: A practical approach to prevent lazy freeriding in P2P networks, in: IEEE International Symposium on Parallel & Distributed Processing, IEEE, 2009, pp. 1–8.

[12] R. Delaviz, N. Andrade, D. Epema, J. Pouwelse, Improved accuracy and coverage in the bartercast reputation mechanism, Proceedings of the 16th annual conference of the Advanced School for Computing and Imaging, Veldhoven, the Netherlands, November 1-3, (2010) 1–8.

[13] C. Barras, File-sharers forced to play fair, http://news.bbc.co.uk/2/hi/technology/6971904.stm (2007 (accessed December 27, 2016)).

[14] D. Gkorou, J. Pouwelse, D. Epema, Trust-based collection of information in distributed reputation networks, in: Proceedings of the 30th Annual ACM Symposium on Applied Computing, ACM, 2015, pp. 2312–2319.

[15] R. Delaviz, J. Pouwelse, D. Epema, Targeted and scalable information dissemination in a distributed reputation mechanism, in: Proceedings of the seventh ACM workshop on Scalable trusted computing, ACM, 2012, pp. 55–66.

[16] P. Otte, Sybil-resistant trust mechanisms in distributed systems (2016).

[17] S. Seuken, D. C. Parkes, Sybil-proof accounting mechanisms with transitive trust, in: Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems, International Foundation for Autonomous Agents and Multiagent Systems, 2014, pp. 205–212.

[18] L. R. Ford, D. R. Fulkerson, Maximal flow through a network, Canadian journal of Mathematics 8 (3) (1956) 399–404.

[19] J. Edmonds, R. M. Karp, Theoretical improvements in algorithmic efficiency for network flow problems, Journal of the ACM (JACM) 19 (2) (1972) 248–264.

[20] Y. Dinitz, Dinitz' algorithm: The original version and Even's version, in: Theoretical computer science, Springer, 2006, pp. 218–240.

[21] A. V. Goldberg, R. E. Tarjan, A new approach to the maximum-flow problem, Journal of the ACM (JACM) 35 (4) (1988) 921–940.

[22] V. King, S. Rao, R. Tarjan, A faster deterministic maximum flow algorithm, in: Proceedings of the third annual ACM-SIAM symposium on Discrete algorithms, Society for Industrial and Applied Mathematics, 1992, pp. 157–164.

[23] J. B. Orlin, Max flows in o (nm) time, or better, in: Proceedings of the forty-fifth annual ACM symposium on Theory of computing, ACM, 2013, pp. 765–774.

[24] R. E. Gomory, T. C. Hu, Multi-terminal network flows, Journal of the Society for Industrial and Applied Mathematics 9 (4) (1961) 551–570.

[25] R. K. Ahuja, T. L. Magnanti, J. B. Orlin, Network flows: theory, algorithms, and applications.

[26] B. L. Crowe, The tragedy of the commons revisited, American Association for the Advancement of Science, 1969.

[27] P. Brussee, Attack-resilient media using phone-to-phone networking (2016).

[28] J. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D. Epema, M. Reinders, M. Van Steen, H. Sips, Tribler: a social-based peer-to-peer system, Concurrency and Computation: Practice and Experience 20 (2) (2008) 127–138.

[29] A. Bakker, R. Petrocco, M. Dale, J. Gerber, V. Grishchenko, D. Rabaioli, J. Pouwelse, Online video using bittorrent and html5 applied to wikipedia, in: 2010 IEEE Tenth International Conference on Peer-to-Peer Computing (P2P), IEEE, 2010, pp. 1–2.

[30] N. Zeilemaker, B. Schoon, J. Pouwelse, Dispersy bundle synchronization, TU Delft, Parallel and Distributed Systems.

[31] R. Huang, T. Polak, Lobster: Limit order book reconstruction system, Available at SSRN 1977207.

[32] R. G. Brown, J. Carlyle, I. Grigg, M. Hearn, Corda: An introduction, R3 CEV, August.

[33] V. Buterin, Transaction spam attack: Next steps, https://blog.ethereum.org/2016/09/22/transaction-spam-attack-next-steps/ (2016 (accessed December 30, 2016)).

[34] B. Viswanath, A. Post, K. P. Gummadi, A. Mislove, An analysis of social network-based sybil defenses, in: SIGCOMM Computer Communication Review, Vol. 40, ACM, 2010, pp. 363–374.

[35] H. Yu, M. Kaminsky, P. Gibbons, A. Flaxman, SybilGuard: defending against sybil attacks via social networks, ACM SIGCOMM Computer Communication Review 36 (4) (2006) 267–278.

[36] H. Yu, P. B. Gibbons, M. Kaminsky, F. Xiao, SybilLimit: A near-optimal social network defense against sybil attacks, in: Symposium on Security and Privacy (SP), IEEE, 2008, pp. 3–17.

[37] G. Danezis, P. Mittal, SybilInfer: Detecting sybil nodes using social networks., in: NDSS, 2009.

[38] I. Keidar, Using social networks to overcome sybil attacks, SIGACT News 42 (2011) 79–101.

[39] N. Tran, B. Min, J. Li, L. Subramanian, SumUp: Sybil-resilient online content voting, in: Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation (NSDI), 2009.

[40] S. Seuken, J. Tang, D. C. Parkes, Accounting mechanisms for distributed work systems, in: Proceedings of the 24th AAAI Conference on Artificial Intelligence, AAAI Press, 2010.