Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

TrustChain: A Sybil-resistant scalable blockchain

Pim Otte, Martijn de Vos *, Johan Pouwelse

Delft University of Technology, Delft, The Netherlands

HIGHLIGHTS

- A tamper-proof, scalable and blockchain-based data structure (TrustChain).
- A Sybil-resistant model to determine trustworthiness (NetFlow).
- A public experiment which addresses freeriding in online communities.

ARTICLE INFO

Article history:
Received 31 December 2016
Received in revised form 24 May 2017
Accepted 25 August 2017
Available online xxxx

MSC:
00-01
99-00

Keywords:
Blockchain
Trust
Reputation
Tamper-proof data structure
Transactions

ABSTRACT

TrustChain is capable of creating trusted transactions among strangers without central control. This enables new areas of blockchain use with a focus on building trust between individuals. Our innovative approach offers scalability, openness and Sybil-resistance while replacing proof-of-work with a mechanism to establish the validity and integrity of transactions.

TrustChain is a permission-less tamper-proof data structure for storing transaction records of agents. We create an immutable chain of temporally ordered interactions for each agent. It is inherently parallel and every agent creates his own genesis block. TrustChain includes a novel Sybil-resistant algorithm named *NetFlow* to determine trustworthiness of agents in an online community. NetFlow ensures that agents who take resources from the community also contribute back. We demonstrate that irrefutable historical transaction records offer security and seamless scalability, without requiring global consensus. Experimentation shows that the transaction throughput of TrustChain surpasses that of traditional blockchain architectures like Bitcoin. We show by using extracted data from a live network that TrustChain has sufficient informativeness to identify freeriders, leading to refusal of service.

1. Introduction

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]. Blockchain technology lets people who have no particular confidence in each other collaborate without relying on a neutral central authority.

As an introduction we will explain the strange transition it has made. Only as recent as 2016 has blockchain technology gone mainstream. The World Economic Forum (WEF) has released an in-depth study in August 2016 [2]. After a year-long investigation of the emerging technology it concludes that the blockchain "will fundamentally alter the way financial institutions do business around the world". Over 80% of banks worldwide participate in blockchain projects in 2017. The Bank of England has been an early supporter of blockchain and stated that this technology "potentially combined with mobile technology, may reshape the mechanisms for making secure payments" [3].

This is an ironic situation since the early promise of blockchain technology was to make banks redundant, bypass central banks, and avoid government regulations. The first generation of blockchain technology, called *Bitcoin*, flourished within the underground economy [4]. Numerous newspaper articles discussed how governments proved to be powerless at stopping online drug trading, facilitated by Bitcoins. However, the FBI confiscated $28.5 million in Bitcoin money from one of the largest online drug dealers and he is now serving a life sentence without the possibility of parole [5]. Governments and banks alike have recently embraced blockchain technology. Rather than staying at the margins of the finance industry, the blockchain is likely to become the beating heart of it. Despite the hype, this technology is still being matured for large-scale usage. This transition is estimated to take at least ten years [2].

The largest team developing blockchain technology is the R3 consortium of 80 global financial firms [6]. Backers of the R3 Corda blockchain include J.P. Morgan, Credit Suisse, Barclays, Bank

* Corresponding author.
E-mail address: m.a.devos-1@tudelft.nl (M. de Vos).

http://dx.doi.org/10.1016/j.future.2017.08.048
0167-739X/© 2017 Elsevier B.V. All rights reserved.

of America, Deutsche Bank, BNP Paribas, and ING. The R3 CEV infrastructure can currently perform a key financial trade function through the blockchain, namely promissory notes, an unconditional promise to pay a determinate sum of money (e.g. commercial papers). R3 CEV can now issue, trade, transfer and redeem promissory notes. In March 2016 the member banks tested five blockchain vendors and three cloud providers, to see how different combinations handled simulated commercial paper transactions.

To conclude, major banks are actively looking into blockchain technology for parts of their core infrastructure.

2. Blockchain architectures

Three distinct blockchain architectures have evolved over time. The architectures in the following list are increasingly more generic and have improved scalability.

- *Permission-less cybercurrency*
  The permission-less aspect of this architecture is a unique property which ensures no middleman needs to be asked for permission, no identity provider needs to approve your application, and no financial entity of any kind is required. One is able to contribute to the processing of transactions without prior involvement in a blockchain. Permission-less cybercurrency such as Bitcoin uses an elegant solution to address the double spending problem: the proof-of-work consensus model. Unfortunately, the requirement of a global, consistent state does not scale and requires additions such as leaders or supervisory servers [7]. As a consequence, Bitcoin only supports roughly seven transactions per second due to limitations in the size of a block. Additionally, while the cybercurrency provides some rudimentary scripting rules, usage in a broader context is limited.
- *Private transaction fabric*
  The word private or permissioned implies that transactions are not exposed to all users by default. Private architectures usually put a single entity or server in control and require complex access control mechanisms to prevent unauthorized users from accessing sensitive data.
- *Permission-less transaction fabric*
  Executing general purpose programs in a permission-less setting while maintaining scalability has proven to be challenging. An example of such a permission-less transaction fabric is Ethereum which focusses on executing smart contracts, code that is invoked by initiating transactions. In 2014 the Ethereum Virtual Machine (EVM) became operational for the first time. Each Ethereum node in the network operates an EVM implementation and executes identical instructions. Like Bitcoin, the scalability of Ethereum is severely limited by the requirement of a consensus mechanism.

All three blockchain architectures aim to facilitate trustworthy transactions at scale.

The problem is that to date, scientists have never managed to design and deploy a self-organizing 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 in permission-less architectures is the Sybil attack, in which adversaries create numerous fake identities to gain a disproportionately large influence [8]. 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 [9], Amazon reviews [10] and Google searches [11]. The requirement for agents to present a trusted identity conflicts with the need for permission-less open membership.

The main contribution of this paper is further broadening of the most generic blockchain architecture devised to date. We present an approach which provides distributed trust, void of any gatekeeper, while still providing strict bounds on the profitability of a Sybil attack. TrustChain is a remarkably simple blockchain-based data structure. It can be used to record transactions and to make these transactions tamper-proof. 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 created an online community of volunteers which share Internet bandwidth and demonstrate the viability of our TrustChain architecture in the context of this community.

The contributions of this work are the following:

- A tamper-proof, scalable and blockchain-based data structure (TrustChain).
- A Sybil-resistant model to determine trustworthiness (NetFlow).
- A public experiment which addresses freeriding in online communities.

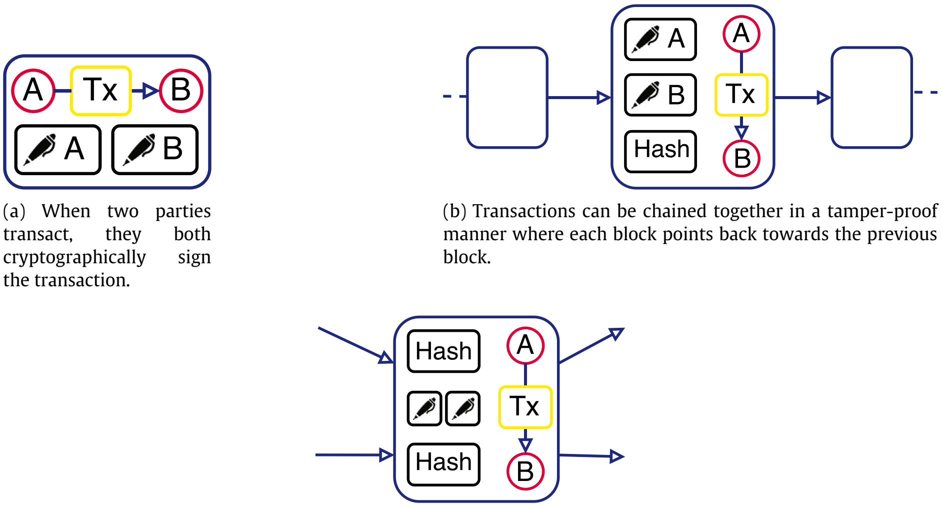
Our work includes both a formal proof of the Sybil resistance of NetFlow and an Internet deployment of TrustChain.

3. 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" [12]. 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 gossipping and has no replication of records since transaction records are only stored by the two or more directly involved parties.

From 2006–2012, there was much excitement in the research community about using social networks to mitigate Sybil attacks [13,14]. Algorithms such as SumUp, SybilGuard, SybilLimit, and SybilInfer provide resistance against Sybils by analysing the social graph [15–18]. In 2007 we designed and Internet-deployed the first *fully* distributed reputation system that prevents lazy freeriding, called BarterCast [19,20]. The BarterCast mechanism calculates reputation of agents by utilizing 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 [21]. The same authors have added the notion of Sybil-proofness and transitive trust to Drop-Edge in subsequent research work [22]. Their work on Sybil-resistant mechanism forms the basis of the NetFlow mechanism described in this paper.

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 [23]. 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 [24]. Alternatively, one can restructure the chain to form a directed acyclic graph of blocks and loosing transaction acceptance rules such that it incorporate transactions even from seemingly conflicting blocks [25]. While these models provide a significant increase in potential speed, they do not allow for unbounded scalability and require complex data structures or consensus mechanisms.



(a) When two parties transact, they both cryptographically sign the transaction.

(b) Transactions can be chained together in a tamper-proof manner where each block points back towards the previous block.

(c) To increase the resistance against tampering, each block also references a block in the chain of the counterparty. This ensures that each block has two incoming and two outgoing pointers.

Fig. 1. Example of the block creation process when agents *A* and *B* have participated in a transaction.

4. TrustChain architecture

The basic idea of the TrustChain architecture is built around the notion of agents transacting with each other. Real-world examples of a transaction include the exchange of files, buying or selling goods and transferring money. Fig. 1(a) represents a transaction between agent *A* and *B*. Each transaction is cryptographically signed by both participating parties using any secure signing mechanism. This ensures that the participation of each user involved in the transaction is irrefutable.

Each participant keeps track of transactions where he was involved in. One way to organize these historical encounters is to chain transactions together in a tamper-proof manner. This idea is illustrated in Fig. 1(b) where the transaction chain of a particular agent in the network is given. Transactions are stored using a blockchain data structure where each block contains one transaction, both signatures of the interacting agents and a pointer to the prior block in the chain. This pointer is often constructed by including a hash value in the block description. Any secure hashing function can be used for this purpose.

The structure in Fig. 1(b) differs from traditional blockchain architectures in a sense that every participant grows and maintains their own chain of transactions. Architectures like Bitcoin or Ethereum maintain one single and global chain containing a trace of all transactions performed by users. Consistency of the chain is guaranteed by a consensus system like proof-of-work. An additional difference is that in traditional blockchains, often multiple transactions are packed together in one block to increase transaction throughput whereas in TrustChain we assume each block describes at most one transaction. After a transaction between two agents has finished, both parties sign the transaction and append a new block to their local chain.

To imply an order on transactions, each element in a blockchain is accompanied by a sequence number $s \in \mathbb{R}$, uniquely identifying the position of a block. The first block in a chain, also called the *genesis block*, is assigned sequence number 1. This sequence number is incremented by one for each subsequent block in the chain.

While a blockchain is an elegant structure to account historical interactions, there is a vulnerability in this approach: the chains of agents are void of any control since each local chain is only maintained by one entity. Transacting agents might decide to not append a transaction to their local chain. The rationale behind this behaviour is that a transaction can be unfavourable for one of the participants, i.e. when a specific agent has only consumed work. In addition, an agent might "rewrite" his local chain by reordering transactions and recomputing prior pointers without much computational effort.

In order to secure the TrustChain architecture against the aforementioned attacks, we include an additional pointer in each block that points back to the last block in the chain of the transaction counterparty. This is presented in Fig. 1(c) which illustrates an element in the chain that is accompanied with two hashes. In TrustChain, each historical encounter has two incoming and two outgoing pointers. Violation of this rule can efficiently be detected, for instance, when creating two blocks with the same prior block pointer (see Section 6.1). The additional pointer to the chain of the counterparty makes it hard to reorder or remove blocks in ones chain since this can be detected by the other party involved in a transaction. This makes that TrustChain can be considered as a mechanism where consensus is reached among participants of a specific transaction instead of consensus on a global level.

As participants are initiating transactions with others, they become quickly intertwined ("entangled") with other users as more blocks are created. TrustChain blocks together form a directed acyclic graph (DAG) of which an example is given in Fig. 2. Each block in the figure is stored in the chains of transacting parties.

In contrast to many existing blockchain architectures which attempt to prevent fraudulent operations like double spending, we aim for a guarantee that fraud can be detected, even after it has been committed already. Not actively preventing fraud allows us to drop the requirement for global consensus while tremendously improving scalability due to the possibility of transactions performed in parallel. We should remark that while consensus is not a necessary element, such a mechanism will add an additional layer of security and verification on top of the TrustChain data structure.

Validation of blocks is performed prior to a block being appended to the local storage of each agent. During validation, the incoming and outgoing pointers, sequence numbers, transaction

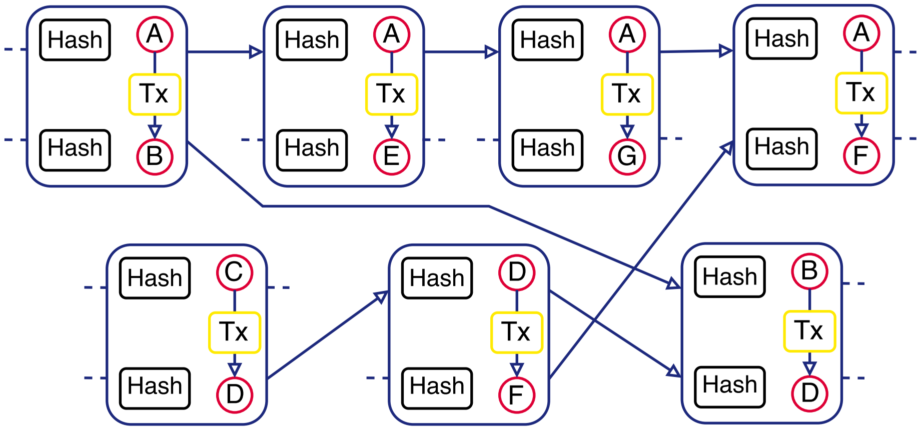


Fig. 2. The tamper-proof TrustChain data structure to record transactions.

data and signatures are verified. Only if a block is marked as valid, it is inserted in the local storage and shared with other network participants.

TrustChain blocks are designed to be exchanged by agents using gossipping and are replicated widely. This makes the system resilient against churn, the phenomena that agents go on- and offline at a fast rate. Each agent operates their own bulk storage of blocks, resulting in partial storage of the global directed graph. Every agent publishes their own unique chain, monitors interactions of others and collects TrustChain data 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 [26,27]. In this work we assume that this specific block collection problem is addressed: we focus on the challenging problem of Sybil-resilient calculation of trust scores and operational systems built upon TrustChain.

5. NetFlow accounting mechanism

We now present the Sybil-resistant NetFlow accounting mechanism that uses the TrustChain graph as input. We consider a distributed network that consists of n agents, each capable of doing work for each other. Recall that we build TrustChain around the notion of agents transacting with each other. In the model discussed in this section, a transaction can be considered as an interaction between multiple agents. Before information about interactions can be exploited, we define an interaction model which includes temporal information about interactions between agents in the network [28].

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 in it
- $w : I \times P \to \mathbb{R}_{\geq 0}$, a function which describes the contribution of an agent in an interaction

Note that $w(i, p) = 0$ must hold if $p \notin a(i)$. The interactions that involve agent p is given by the following totally ordered set.

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

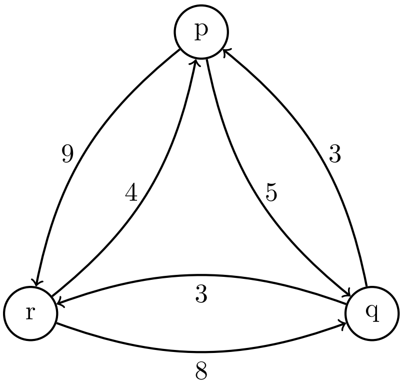


Fig. 3. An interaction graph involving three agents. In this graph, agent p contributed a total of 9 units work to agent r.

An interaction involves two distinct 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 necessity is that something induces an order on the set of interactions of each agent. In the case of TrustChain, this order is induced by the time at which a transaction is signed and stored.

The definition of an ordered interaction model can be applied directly to the TrustChain data structure presented in Section 4. Each interaction between two agents is backed by a record in the data structure.

Interactions are often represented by a graph, which motivates the definition of an interaction graph.

Definition 2 (*Interaction graph*). Let $M = \langle P, I, a, w\rangle$ be an ordered interaction model. The weighted interaction graph $G_M = (V, E, w)$ is defined as follows:

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

Let $(v_p, v_q) \in V$, then the weight w of (v_p, v_q) is equal to the sum of all contributions that involve agent p.

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

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. An example of an interaction graph is given in Fig. 3. Note that the definition of an interaction graph does not

Please cite this article in press as: P. Otte, et al., TrustChain: A Sybil-resistant scalable blockchain, Future Generation Computer Systems (2017), http://dx.doi.org/10.1016/j.future.2017.08.048.

reflect the notion of time in any way and defines interaction on a more global level.

While the interaction graph represents interactions on a network-wide scale, it is realistic to assume that individual agents do not have knowledge about all interactions but are only interested in a subset of them. The following definitions are inspired by the work of Seuken and Parkes [22].

Definition 3 (*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, where the set of interactions in the model is a subset of all interactions that includes at least all interactions of agent *i*.

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

We assume that each agent is interested in contributing some work to other agents. The set of such agents is defined to be the choice set.

Definition 4 (*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 assign a score to each agent in a choice set using an accounting mechanism.

Definition 5 (*Accounting mechanism*)**.** An accounting mechanism M takes as input a subjective work 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 5 defines what is basically a scoring mechanism. This mechanism can be used to select an agent from the choice set that receives a unit of work, according to a specific allocation policy.

Definition 6 (*Allocation policy*)**.** Given subjective work graph G_i, choice set C_i and an accounting mechanism M, an allocation policy $A : \mathbb{R}^n \to P$ is a function that maps a set of agent scores $S^M(G_i, C_i)$ to an agent $j \in C_i$. This agent is chosen to receive a unit of work from agent *i*.

The used allocation policy often depends on the type of application. A basic allocation policy could be to choose the agent with the highest score and select an agent randomly in case of a tie breaker. This policy is also referred to as Winner-Take-All (WTA).

We now present the *NetFlow* accounting mechanism that is used to assign a score to agents in a distributed network.

Definition 7 (*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 value for each agent $j \in C_i$.

$$c_j = \max\{MF_{G_i}(j, i) - MF_{G_i}(i, j), 0\}. \tag{3}$$

In Eq. (3), $MF_{G_i}(i, j)$ denotes the value of the maximum flow (maxflow) from *i* to *j* in G_i, where the weights are the capacities on the edges.

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.

$$S_j^M(G_i, C_i) = MF_{G_i^N}(j, i). \tag{4}$$

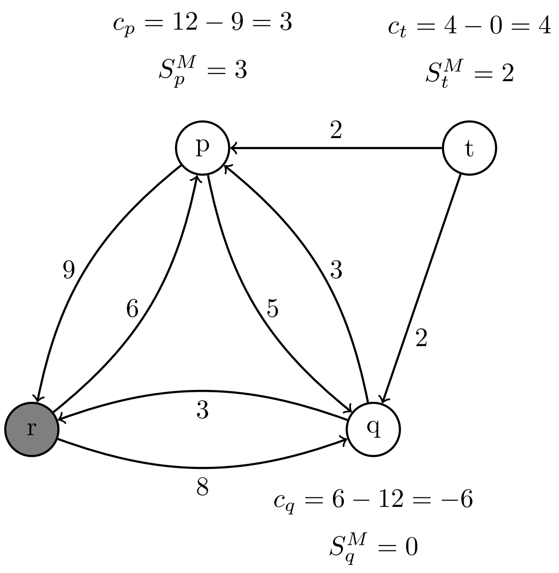


Fig. 4. A NetFlow computation performed from the perspective of agent *r*. The node capacities *c* and assigned scores S^M are indicated for each agent in the subjective work graph.

An example of a NetFlow computation performed from the perspective of agent *r* is given in Fig. 4. This subjective work graph G_r contains four agents. The first step of the NetFlow algorithm is determining node capacities for each agent in the network. In the following explanation, we calculate the final trustworthiness score assigned to agent *p*. The maximum flow from agent *p* to agent *r* is equal to 12 and the maximum flow from agent *r* back to agent *p* is equal to 9, hence the node capacity of *p* (c_p) becomes $\max(12 - 9, 0) = 3$.

Next, we determine the score S_p^M as assigned by the NetFlow accounting mechanism of agent *p* which is equal to 3. Note that at most 3 units of work can flow through agent *p* in the subjective work graph. This process is repeated to determine the scores of other agents in the network.

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

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. We scale NetFlow by rating work performed by the calculating agent *i* lower than other work, which is equivalent to rating all work by other agents higher. This leads to the following definition of NetFlow where the value of work by the calculating agent is scaled by a factor α.

Definition 9 (*α-NetFlow limited contribution*)**.** Given a subjective work graph $G_i = (V_i, E_i, w_i)$, a choice set C_i and $\alpha \geq 1$, then the α-scaled weights $w_{i,\alpha}$ are defined as follows:

$$w_{i,\alpha}(e) = \begin{cases} \dfrac{w_i(e)}{\alpha} & \text{if } e = (i, j) \text{ with } j \in V_i \\ w_i(e) & \text{otherwise.} \end{cases} \tag{5}$$

Please cite this article in press as: P. Otte, et al., TrustChain: A Sybil-resistant scalable blockchain, Future Generation Computer Systems (2017), http://dx.doi.org/10.1016/j.future.2017.08.048.

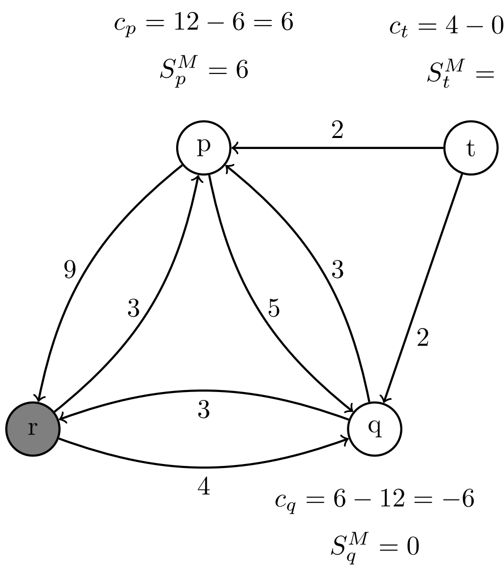


Fig. 5. A 2-NetFlow computation performed from the perspective of agent r. The node capacities c and assigned scores S^M are indicated for each agent in the subjective work graph.

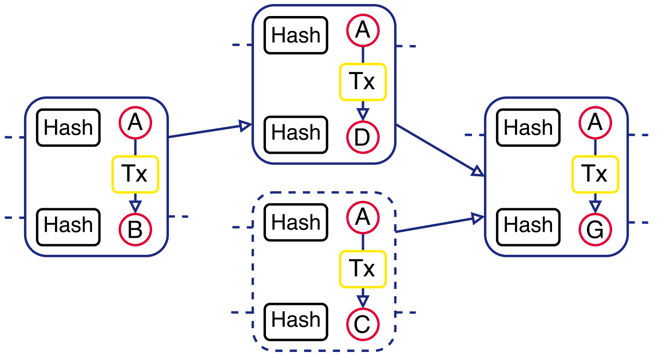


Fig. 6. An example of a *double spending attack* performed by agent A. Two blocks with the same prior pointer are created. Agent A wishes to hide the dashed block.

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

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

An example of a α-NetFlow computation is given in Fig. 5. It has a similar, but slightly weaker defence against Sybil attacks than NetFlow which will be discussed in Section 6.3.

6. Security analysis

We now elaborate some attacks on the TrustChain architecture where a malicious agent attempts to gain an unfair advantage, either by tampering with the data structure described in Section 4 or by abusing the NetFlow accounting mechanism discussed in Section 5. While the overview presented here is likely not exhaustive, it highlights some vulnerabilities that are inherent to blockchain architectures and accounting mechanisms.

6.1. Double spending attack

Similar to Bitcoin it is possible to *fork* your own blockchain by creating two different transaction branches. TrustChain has a defense mechanism against this key attack, violators will be detected and lose the trust of others.

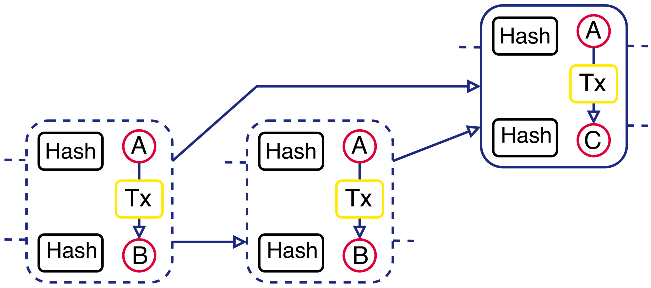


Fig. 7. An example of a *replay attack* performed by agent M. The dashed blocks contain the same transaction data and signatures.

The attack is illustrated in Fig. 6 where the TrustChain structure of an agent performing a double spending attack is presented. In this scenario, agent A wishes to hide the dashed block, the interaction with agent C and only propagates information about his interaction with agent D. While this attack might seem successful at first, the hidden transaction with agent C will eventually be detected when an additional agent, say B, learns about the historical encounters of agent C. During verification of the chain of C, the transaction that A wishes to hide is discovered. This contradicts the knowledge of B about transactions in which agent A has been involved. The two blocks that agent A has created together form a *proof-of-fraud* and should be broadcast in the network. The proof can be used by other agents to verify the fraudulent action with little computational effort, leading to blacklisting or refusal of service.

6.2. Replay attack

Where the double spending attack is more involved with hiding transactions of an agent, a replay attack attempts to reuse the transaction signature created by the counterparty, essentially replaying a transaction. A malicious agent reuses the pointer to a prior block of the other party. The replay attack is illustrated in Fig. 7 where agent A grows his chain with the same transaction twice. The motivation behind this attack is that a malicious agent can claim that he has been involved in a transaction that is beneficial for him multiple times. This attack is relatively easy to discover: when verifying correctness of the transaction chain of agent A by another agent, he detects when there are two blocks that have the same outgoing pointer. The proof-of-fraud consists of the blocks created by the malicious agent during the replay attack; any agent in the network can verify the fraud by observing the outgoing pointers of the blocks at issue.

6.3. Sybil attack

A successful Sybil attack increases the reputation of some agents or lowers the reputation of others by initiating interactions in the network. This attack is formally defined in the following definition.

Definition 10 (*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 $e \in E_s$.

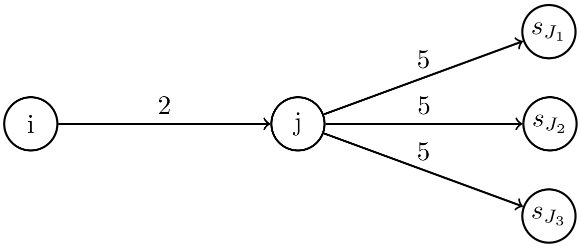


Fig. 8. A Sybil attack performed by agent j. This agent creates three Sybils which he uses to boost his contributions.

A Sybil attack is presented in Fig. 8 where agent j creates three Sybil identities and performs work for them in order to boost his own contributions. This is only beneficial for the attacker as long as he receives more work from others than the amount of work that he has to perform. We formalize this profitability in the next definition.

Definition 11 (*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 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:

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

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".

We now prove that the profitability of a Sybil attack on NetFlow is bounded.

Theorem 1 (*Sybil-resistance of NetFlow and α-NetFlow*). *NetFlow and α-NetFlow are resistant against weakly beneficial Sybil attacks.*

Proof. 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 node capacity of some agent in J that has directly contributed to the network must drop by 1 unit. 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:

$$\sup \left\{ \frac{\omega_+^n}{\omega_-^n} : n \in \mathbb{N}, \omega_-^n \neq 0 \right\} \leq 1. \tag{7}$$

This shows that the NetFlow accounting mechanism is resistant against weakly beneficial Sybil attacks with a profit no more than 1.

Since NetFlow is resistant against weakly beneficial Sybil attacks with profit at most 1 and the only difference between NetFlow and α-NetFlow 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 involves agent 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 NetFlow, which still implies an upper bound on the Sybil attack profit and makes α-NetFlow also resistant against weakly beneficial Sybil attacks. □

6.4. Other attacks

We conclude this section with an overview of various other attacks.

Hiding Blocks — Once agents start interacting in the network, records are created. An agent might want to only expose transactions that have a positive influence on his reputation while hiding those that decrease the standing of this particular agent. The TrustChain data structure defends against this attack: since every record contains a sequence number, anyone in the network can request specific records of others, which agents cannot refuse to provide without being detected as a fraud. If an agent is unwilling to provide his historical encounters, one might choose to not transact with this agent until the requested records are provided and verified. Note that an agent cannot prevent his transactions from being propagated to other users by the counterparty.

Refusal to Sign— A malicious agent can decide to not sign a transaction that is not in his favour. Mitigating this attack is far from trivial since no agent can be forced to sign a transaction. A possible recourse is to not interact with this agent again. Another defence mechanism is to gradually build trust between participants by splitting the transactions in smaller amounts. In this situation, if an agent refuses to sign a transaction, the interaction is aborted. Such a scheme reduces the reputation at stake when a counterparty initiates this attack at the cost of more overhead due to the creation and storage of additional records.

Whitewashing— The permission-less nature of TrustChain enables the creation of additional identities at any time without much effort. If an agent suffers from a bad reputation, he can simply get rid of his current identity and take on a new one. This process is also called *whitewashing*. It is not desired to refuse services to identities that have yet to build a reputation in the network since that withholds users from joining the network at all. An adequate solution for this problem is to prioritize new identities lower by the allocation policy defined in Definition 6.

7. Performance analysis

In the first part of this section, we quantity the transaction throughput of the TrustChain data structure. Next, NetFlow is applied to a real-world scenario and we show how it can be used to refuse services to freeriders in online communities.

7.1. Transaction throughput

The ability to record transactions in an efficient, light-weight, and scalable manner is key. Recall that TrustChain transactions are tamper-proof irrefutable records, cryptographically signed by both participants as described in Section 4. We now aim to quantify the overhead and scalability of TrustChain, both on a personal computer and within a challenging environment with constrained computational resources: mobile devices.

We fully implemented TrustChain in the Python programming language and created a simple application that creates TrustChain transactions, all released as open source[1] [29]. For this experiment, we setup two agents on the same device that are interacting with each other. Each transaction is cryptographically signed by both parties before being inserted in the local storage of the agents.

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

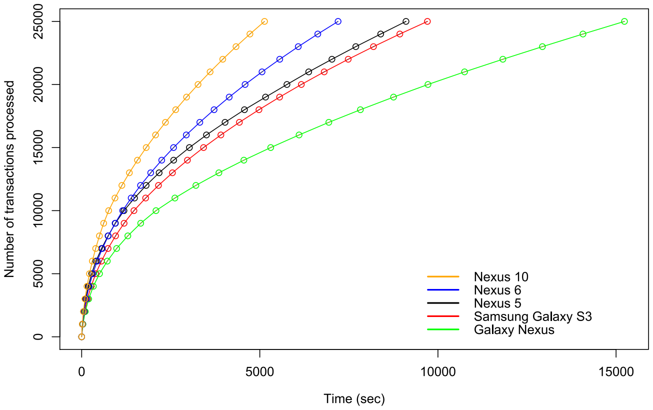


Fig. 9. Transaction throughput on various mobile Android devices.

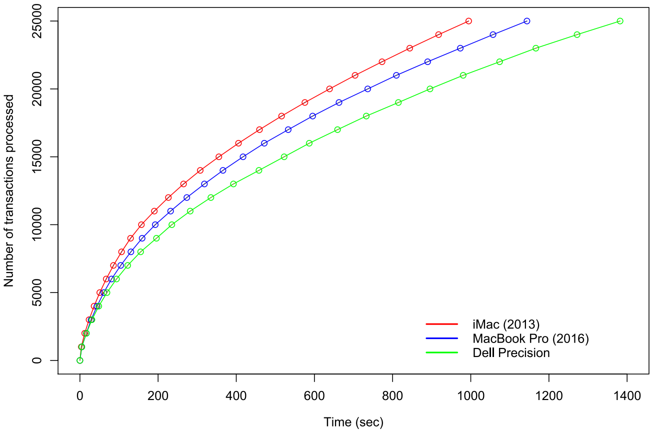


Fig. 10. Transaction throughput on various personal computers.

Fig. 9 shows the required time to create and store 25 000 TrustChain transactions on various types of mobile devices. Fig. 10 illustrates the transaction throughput during same experiment, performed on personal computers. The horizontal axis denotes the time into the experiment in seconds while the vertical axis specifies the number of TrustChain transactions that have been processed. The experiment performed on mobile devices uses five Android-based devices to demonstrate the usability of TrustChain on low-cost and aged hardware, namely a Nexus 10 tablet (2012), a Samsung Galaxy S3 (2012), a Nexus 5 (2013), a Nexus 6 (2014) and a Samsung Galaxy S6 (2015). For the experiment that considers personal computers, we used an iMac (2013), a Macbook Pro (2016) and a Dell Precision M4600 laptop. At the start of this experiment the durable storage is empty and new transactions are generated and inserted quickly in the embedded SQLite database. With subsequent database growth the insertion overhead somewhat increases. The most likely explanation for this behaviour is that each insertion requires a database query to fetch information about the latest block of a specific agent, an operation that slows down as the size of the database increases.

Fig. 9 shows that the transaction throughput of a Nexus 10 tablet surpasses that of other mobile devices with an average speed of 4.9 transactions per second. This can be attributed to the better hardware the device comes with. The slowest mobile device, the Galaxy Nexus, has an average throughput of 1.6 transactions per second. An interesting observation is that the first 1000 transactions are processed quickly, with an average of 54.8 transactions per second on the Nexus 10 and 28.7 transactions per second on the Galaxy Nexus.

The throughput on personal computers presented in Fig. 10 is an order of magnitude higher compared to the performance on Android devices. The slowest personal computer we used during this experiment, the Dell Precision M4600, achieves an average transaction throughput of 18.1 transactions per second whereas the first 1.000 records are created with an average speed of 210.3 transactions per second.

These results show that even on ageing mobile hardware it is possible to process thousands of transactions easily. In addition, creating and retrieving records for hours continuously is also affordable. Our experiment does not include network latency required to disseminate records to the counterparty of each transaction, which should negatively impact the observed throughput rate. The code in its current form is not optimized for performance. We conclude that *all* current Bitcoin transactions can be handled by TrustChain using a single Android smartphone if creation of trust and global consensus did not require a computational intensive mechanism.

7.2. NetFlow performance

We now focus on the performance of the NetFlow accounting mechanism discussed in Section 5, both from a theoretical and practical point of view.

7.2.1. Computational time complexity

In order to compute the trustworthiness score of one other agent with NetFlow, up to $2n + 1$ max-flow computations are necessary, where n is the number of agents in a choice set. In case one would compute the scores of all other agents, $3n$ 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, n denotes the number of agents and m denotes the number of edges in the network.

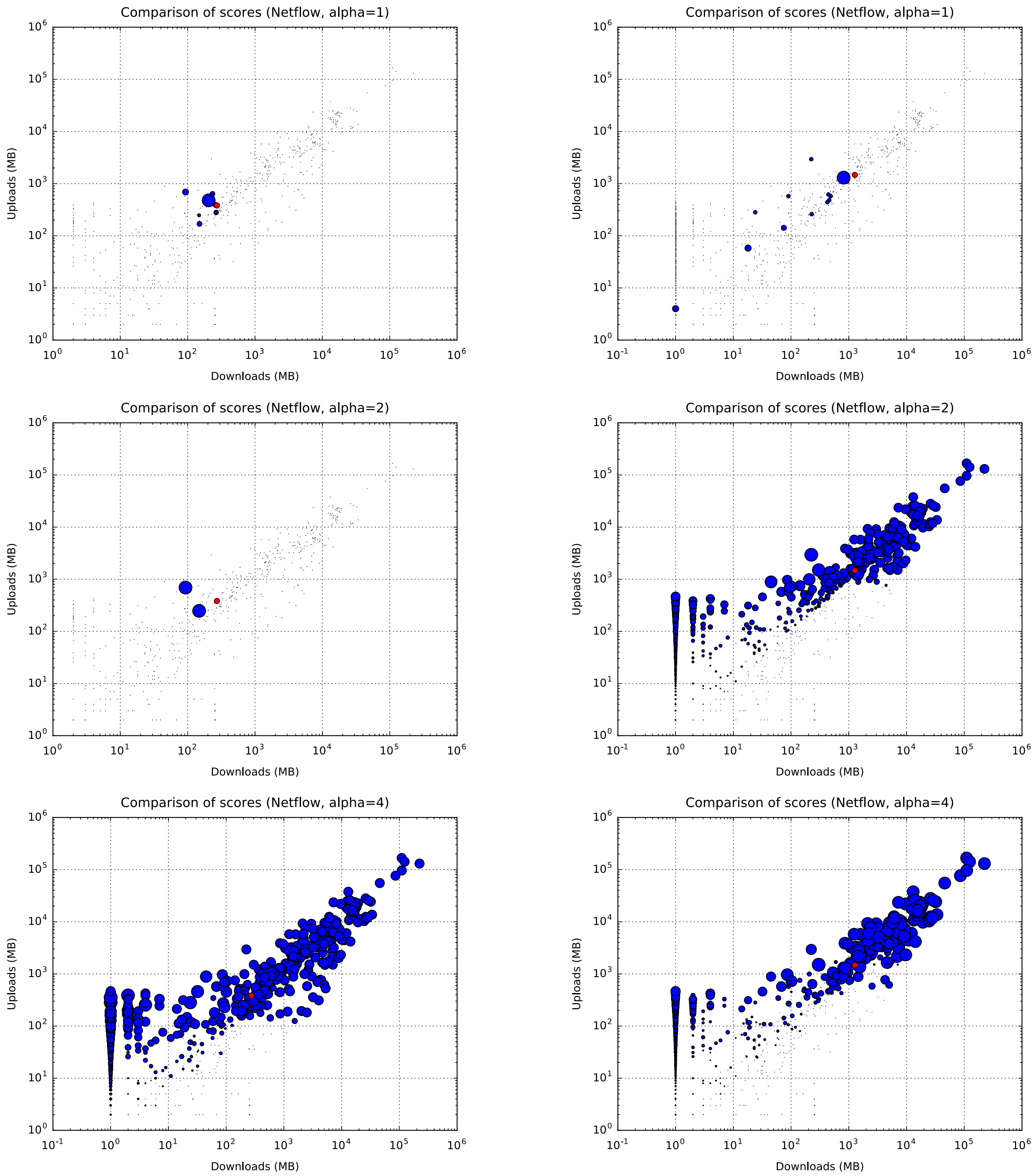
Well-known algorithms for max-flow calculation include Ford–Fulkerson, with computational complexity $O(m|f|)$, where $|f|$ is the magnitude of the maximum flow [30]. This is a pseudo-polynomial algorithm, since the complexity depends on the magnitude of numbers in the instance. The Edmonds–Karp algorithm specifies the order in which augmenting paths are considered, namely by doing a breadth-first search [31]. 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 [32]. Goldberg and Tarjan introduced the preflow-push algorithm [33] 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 $O(nm\log_{\frac{m}{n\log(n)}} n)$ [34]. In combination with work by Orlin, this results in an $O(nm)$ algorithm for max-flow [35]. 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 [36], however, this method only works for undirected graphs (the interaction graph as described by Definition 2 is a directed graph). 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 [37]. This work is from 1993, and to the best of 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.

Please cite this article in press as: P. Otte, et al., TrustChain: A Sybil-resistant scalable blockchain, Future Generation Computer Systems (2017), http://dx.doi.org/10.1016/j.future.2017.08.048.



(a) Computing agent contributed and consumed around 120 MB.

(b) Computing agent contributed and consumed around 1050 MB.

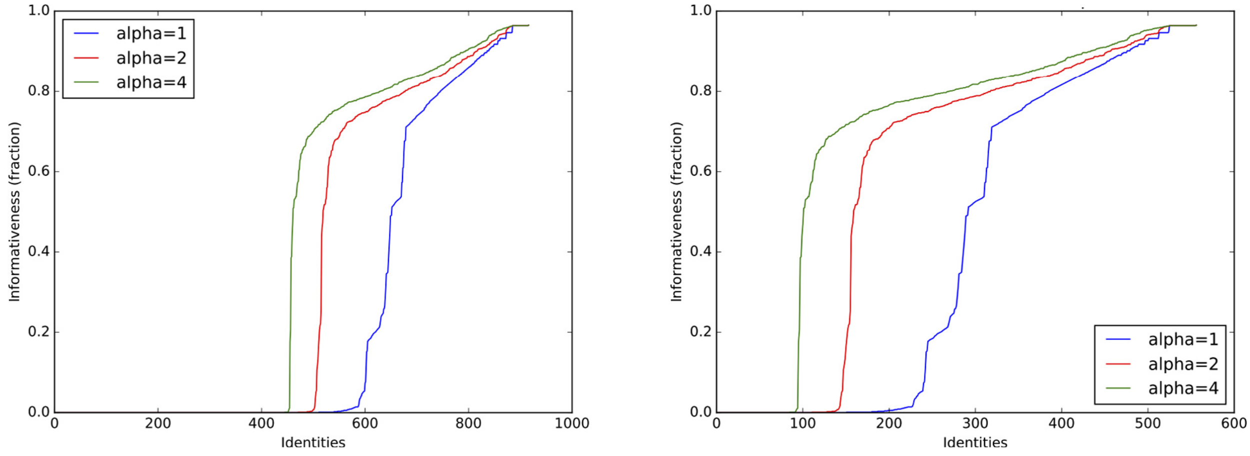
Fig. 11. The NetFlow trustworthiness scores from the perspective of several agents. The size of bubbles indicate the assigned scores (different scales across plots). The red bubble specifies the computing agent. For each agent, we consider NetFlow computations with $\alpha = 1$, $\alpha = 2$ and $\alpha = 4$.

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 $O(n^3m)$.

7.2.2. Real-World deployment and experimentation

Next, we focus on the evaluation of the Sybil-resistant NetFlow mechanism. We have conducted a public experiment in which 917 volunteers recruited from the Internet participated in our open one-month study. These volunteers installed and used our YouTube-like video-on-demand platform, called Tribler [38]. This software is part of our long running “bandwidth-as-a-currency” research line, with a first operational system deployed in August 2007 [39]. In prior experimental work we collaborated with Wikipedia.org and enabled bandwidth donations to their website [40].

Each of our volunteers operated a TrustChain implementation, created his own genesis block upon installing our software, and automatically advertised this publicly on the network. The 917 discovered genesis blocks 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 application users pool their



(a) All agents.

(b) Agents who have downloaded data.

Fig. 12. Informativeness curves for $\alpha = 1, \alpha = 2$ and $\alpha = 4$.

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 as a transaction. We extracted a TrustChain dataset from the Tribler network.

Fig. 11 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 respectively. For each volunteer we calculate NetFlow-based trustworthiness scores using the mechanism given in Definition 7 and 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). The agent whose point of view is taken for the computation is marked red. 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.

We first consider the influence of the NetFlow scaling parameter α and how informative the mechanism explained in Section 5 is. Recall that if we increase the value of α, we value contributions from other agents higher. Upon inspection of Fig. 11, several interesting observations can be identified. Increasing the scaling factor α does indeed increase the informativeness of the mechanism, resulting in more non-zero scores for 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 contribution of the agent that performs computation of NetFlow scores. This effect decreases significantly when we increase α: 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 values of 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 accurately yield a ranking. Fig. 12 provides informativeness curves with different values for α where the leftmost curve corresponds to a higher value of 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, Fig. 12(b) shows the same data, excluding agents that have not downloaded any data. Observe that as α increases, so does the informativeness since more agents are assigned a positive score. Furthermore, for a higher value of α the set of agents with zero informativeness decreases in size. Also note that there is quite a sharp jump from 0 informativeness to around 0.7.

8. Conclusions

We demonstrated the viability for a new direction in blockchain research by proposing a generic method to create trust. We enable new areas of blockchain usage, centred around the notion of trusted transactions. Our TrustChain work uses tamper-proof, temporal ordered and cryptographically signed transaction records to create irrefutable proof of past interactions. 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 contributions to agents who have been helpful to others in the past.

TrustChain in general is designed to be a scalable solution, in the current Python-based unoptimized form capable of processing around 210 transactions per second using modern hardware. The architecture avoids the double spending problem and does not require proof-of-work mechanisms, global transaction broadcasts, leadership elections, permissions, sharding or a central authority.

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

Experimental results indicate that TrustChain is capable of freerider identification in online communities 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 classify agents based on contributions.

It is straightforward to extend the data structure to support transactions between multiple parties by adding additional ingoing and outgoing pointers to the chains of all involved agents in the transaction. When we consider a transaction between *n* parties, each transaction block would contain *n* incoming and *n* outgoing pointers. Likewise, each transaction would be cryptographically signed by *n* entities.

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 a consensus model under development.

References

[1] T. Economist, (2015) The promise of the blockchain: the trust machine (accessed December 30, 2016). http://www.economist.com/news/leaders/21677198-technology-behind-bitcoin-could-transform-how-economy-works-trust-machine.
[2] World Economic Forum, The future of financial infrastructure (2016).
[3] Bank of England, One bank research agenda (2015).
[4] S. Nakamoto, Bitcoin: A peer-to-peer electronic cash system.
[5] A. Greenberg, FBI says it's seized 28.5 million in Bitcoins from Ross Ulbricht, alleged owner of silk road 2013 (accessed may 19, 2017). URL https://www.forbes.com/sites/andygreenberg/2013/10/25/fbi-says-its-seized-20-million-in-bitcoins-from-/ross-ulbricht-alleged-owner-of-silk-road/#64892fa42765..
[6] The R3 consortium, http://www.r3cev.com, accessed: 2017-05-19.
[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] J.R. Douceur, The sybil attack, in: 1st International Workshop on Peer-To-Peer Systems (IPTPS), Springer, 2002.
[9] P. Resnick, R. Zeckhauser, J. Swanson, K. Lockwood, The value of reputation on eBay: A controlled experiment, Experimental Economics 9 (2) (2006) 79–101.
[10] 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.
[11] L. Page, S. Brin, R. Motwani, T. Winograd, The PageRank citation ranking: bringing order to the web.
[12] R.G. Brown, J. Carlyle, I. Grigg, M. Hearn, An introduction, R3 680 CEV, August (2016).
[13] B. Viswanath, A. Post, K.P. Gummadi, A. Mislove, An analysis of social network-based sybil defenses, in: SIGCOMM Computer Communication Review, 40, ACM, 2010, pp. 363–374.
[14] I. Keidar, Using social networks to overcome Sybil attacks, SIGACT News 42 (2011) 79–101.
[15] H. Yu, M. Kaminsky, P.B. Gibbons, A. Flaxman, SybilGuard: defending against Sybil attacks via social networks, ACM SIGCOMM Computer Communication Review 36 (4) (2006) 267–278.
[16] 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.
[17] G. Danezis, P. Mittal, SybilInfer: Detecting Sybil Nodes using Social Networks., in: NDSS, 2009.
[18] 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.
[19] M. Meulpolder, J.A. Pouwelse, D.H.J. Epema, H.J. 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.
[20] 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, (ISSN: 978-90-79982-08-0) (2010) 1–8.
[21] 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.
[22] 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.
[23] 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.
[24] 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.
[25] Y. Lewenberg, Y. Sompolinsky, A. Zohar, Inclusive block chain protocols, in: International Conference on Financial Cryptography and Data Security, Springer, 2015, pp. 528–547.
[26] 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.
[27] 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.
[28] P. Otte, Sybil-resistant trust mechanisms in distributed systems, 2016.
[29] P. Brussee, Attack-resilient media using phone-to-phone networking, 2016.
[30] L.R. Ford, D.R. Fulkerson, Maximal flow through a network, Canad. J. Math. 8 (3) (1956) 399–404.
[31] J. Edmonds, R.M. Karp, Theoretical improvements in algorithmic efficiency for network flow problems, J. ACM 19 (2) (1972) 248–264.
[32] Y. Dinitz, Dinitz' algorithm: the original version and even's version, in: Theoretical Computer Science, Springer, 2006, pp. 218–240.
[33] A.V. Goldberg, R.E. Tarjan, A new approach to the maximum-flow problem, J. ACM 35 (4) (1988) 921–940.
[34] 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.
[35] 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.
[36] R.E. Gomory, T.C. Hu, Multi-terminal network flows, Journal of the Society for Industrial and Applied Mathematics 9 (4) (1961) 551–570.
[37] R.K. Ahuja, T.L. Magnanti, J.B. Orlin, Network flows: theory, algorithms, and applications.
[38] 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.
[39] C. Barras, File-sharers forced to play fair, http://news.bbc.co.uk/2/hi/technology/6971904.stm(2007 (accessed December 27, 2016)).
[40] 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.

Pim Otte received his M.Sc. title from the Delft University of Technology in 2016. His research focussed on the trust problem in distributed systems where he proposed and investigated two Sybil-resistant algorithms that can be used in online communities to prevent freeriding.

Martijn de Vos is a Ph.D. student at Delft University of Technology where he performs research on a blockchain-regulated decentralized and scalable market. In 2016, he received his M.Sc. degree and did research on technical debt in complex distributed software systems. He is also one of the lead developers of the Tribler video-on-demand client, installed by 1.8 million people over the past decade.

Johan Pouwelse is an associate professor at Delft University of Technology, specialized in large-scale cooperative systems. During his Ph.D. he created the first system for cooperative resource management for portable devices on Linux. This resulted in the first portable hardware and Linux driver capable of reducing CPU frequency and voltage from user-space in 2001. The driver got accepted into the Linux kernel and this architecture is still used by every Android and iOS device. Also, he conducted the first resource usage measurements for IEEE 802.11b, known now as wifi. After receiving his Ph.D., he conducted one of the largest measurements of the Bittorrent P2P network. He founded the Tribler video-on-demand client in 2005, it has been installed by 1.8 million people over the past decade. Tribler serves as a living laboratory and proving ground for next generation self-organizing systems research and ledger technology. In 2007 his research team launched BarterCast, a primitive ledger. Dr. Pouwelse has (co-)authored over 129 scientific papers.

Please cite this article in press as: P. Otte, et al., TrustChain: A Sybil-resistant scalable blockchain, Future Generation Computer Systems (2017), http://dx.doi.org/10.1016/j.future.2017.08.048.