A Structural Analysis of Bitcoin

Clemens H. Cap

Department of Computer Science
University of Rostock
Albert Einstein Strasse 22
D-18059 Rostock, Germany
clemens.cap@uni-rostock.de

Abstract: There is no formal framework for describing the core structural concepts of Bitcoin or for attempting a correctness proof of the algorithm. This contribution presents several elements which may serve as building blocks. A distributed model for describing the states enclosed in a Bitcoin network is provided. Concepts for modeling the swarm behavior of Bitcoin are analyzed.

1 Introduction

The probably most famous and true quote on Bitcoin has been coined by DAN KAMINSKY in [Kam11]: *The first five times you think you understand it, you don't.*

Started on Januar 3, 2009, the Bitcoin peer-to-peer network has grown very quickly. It has been run from 739,033 unique IP addresses over a three month period[1] and currently has between 20,000 and 60,000 running nodes at every moment[2]. So far, it has proven astonishingly stable and resilient against actual and paper-and-pencil attacks. This success is caused by a wealth of advanced cryptographic and networking concepts. Several incidents with stolen Bitcoin sums, from which the layman could get a different opinion, were due to security issues on the hosting server and misunderstandings by the user; they are not caused by flaws of Bitcoin. Unfortunately, beyond the self-published white paper [Nak09] of the pseudonym SATOSHI NAKAMOTO and a few general rules, message formats and posts on `bitcoin.org`, there is no formal approach to Bitcoin from a conceptual point of view, no specification and no correctness proof. Moreover, much conceptual elegance is hidden inside of the C++ reference client.

This contribution attempts a structural analysis of some Bitcoin concepts. It is not meant as formal specification and does not reflect all essentials of the algorithm, but contains what the author regards as core building blocks[3]. The text is not meant as introduction; a general understanding of Bitcoin is assumed. The intention is to ultimately develop a formal foundation for Bitcoin, which is formally more precise, more abstract and more

[1] http://blogs.umb.edu/williamfleurant001/2012/01/13/bitcoin/
[2] https://en.bitcoin.it/wiki/Bitcoin_Map
[3] during what might qualify as his fourth attempt in understanding Bitcoin – as far as DAN KAMINSKY's quote is concerned.

flexible than the currently available descriptions. There is hope that this may help to a better understanding and in the further evolution of the current algorithm.

2 A World of States and Transactions

2.1 Centralized Scenario

Imagine a world, the *state* of which is described by a key-value store, registry, or more complex algebraic data structure [Wir90]. Examples are a bank (i. e. a key-value store of account numbers with balances), a domain name system (i. e. a list of domain names with IP-addresses) or an expert system (i. e. a consistent list of axioms and rules from a logical system). The state of this world initially is empty (i. e. there are no accounts or domain names; the list of facts and rules is empty). *Transactions* may change the state (i. e. by creating or deleting accounts and transferring money; by CRUD-ing DNS-type name resolutions; by modifying the list of logical formulae). Not every transaction may be applied in every state (eg. if we transfer money from account $\mathfrak{a}$ to $\mathfrak{b}$, the state in which we apply this transaction must know accounts $\mathfrak{a}$ and $\mathfrak{b}$ and $\mathfrak{a}$ must have a sufficient balance).

We model this by a set $\mathcal{S}$ of states and $\mathcal{T}$ of transactions, and a function[4] $\nu : \mathcal{T} \times \mathcal{S} \to \mathcal{S} \oplus \{\dagger\}$. If transaction $t \in \mathcal{T}$ is performed on state $s \in \mathcal{S}$, the resultant state is $\nu(t, s) \in \mathcal{S}$. If, however, $\nu(t, s) = \dagger$, then transaction t *cannot be applied* to state s. Two transactions t_1 and t_2 *commute on state* s, iff $\nu(t_1, \nu(t_2, s)) = \nu(t_2, \nu(t_1, s))$ and all involved states differ from $\dagger$.

2.2 Distributed Scenario

A *distributed* scenario consists of a number of participants, each of which has a *(local) state*, accessible only to the participant. The participants communicate asynchronously (i. e. communication takes a certain, usually predictable, amount of time).

Initially, the local states are *empty* and all participants share the illusion of an identical (empty) world. Soon the participants start issuing transactions and change their own local state (eg. they open accounts and deposit money; they invent domain names and create address-mappings; they describe objects by axioms, provide rules for reasoning, and apply these rules). Thus the local states of our participants become inconsistent. For example, in the state of ALICE a bank account might have a different balance than in the state of BOB and the DNS-system might resolve domain name `eg.com` for ALICE and for BOB to different addresses. A consistent unification of the individual local states to a single (virtual) global state no longer exists. It is no longer possible to maintain the illusion of a common world, of which the local states are (partial, time-delayed) views. However, cooperative applications (such as banks, domain name systems, expert systems, but also human communication) need this illusion: A distributed software layer must therefore reproduce the artifact of a globally consistent state in order to be considered correct.

In Bitcoin, participants issue transactions to change their local state and broadcast transactions to other participants using *gossiping* [EFLF07]: A node does not send its transactions to all other participants but only to a subset to which it is connected; recipients forward

[4] The sum operator $\oplus$ denotes the *disjoint* union.

received transactions to their own peers unless they already knew the received transaction themselves. Broadcast is *not reliable* (i. e. a number of transactions may not be delivered to a number of nodes) and *not ordered* (i. e. the sequence in which transactions are received may differ from node to node and from the sequence in which they were sent). This is consistent with the failure model of Bitcoin, where a number of Byzantine[5] nodes may exhibit completely arbitrary or malicious behavior, and with the state of the art in totally ordered broadcast [DSU04] (impossible in asynchronous models with crash failures, difficult in Byzantine settings or without additional assumptions, such as channel reliability or trustworthy supernodes [RJ08], scaling badly if these assumptions are dropped).

This produces problems: A node may receive transaction t_2 in a state where it cannot be applied (eg. one node generates a bank account by t_1 and then references it in t_2; a different node receives t_2 before t_1). In spite of unreliable broadcast, every transaction should eventually be recognized by every node. If asked for their current state, different participants may report different states: There is no notion of global time and no mechanism to have the participants report their state at the "same" time. The sequence of states seen by a participant may depend on the participant. All this is due to the following reasons:

Latency: Some participants have not yet received all issued transactions or have not yet applied them for updating their local state.

Unreliable Broadcast: Some participants did not receive a transaction.

Order of Transactions: Two participants X and Y may have seen two transactions t_a and t_b in a different order. This may produce three conceptually different effects:

(1) *Same final state, different path:* X and Y are in the same (local) state s_0. X sees the sequence $t_a t_b$ and Y sees the sequence $t_b t_a$. Assume that t_a and t_b commute on state s_0. This guarantees an identical final state $\nu(t_a, \nu(t_b, s_0)) = \nu(t_b, \nu(t_a, s_0))$, but the intermediary states, $\nu(t_a, s)$ at X and $\nu(t_b, s)$ at Y, may differ.

(2) *Different final states:* In addition to case (1), iff t_a and t_b do not commute on state s_0, the final states at X and Y will be different.

(3) *Different sequence:* X and Y are in the same state s_0. Both see an infinite stream[6] of the same set $\{t_1, t_2, t_3, \ldots\}$ of transactions, but in different interleavings. Even if we assume that *every pair* of transactions t_i, t_j commutes on *every* reachable state s, it is possible that the common starting state s_0 is the *only state* the sequences at X and Y have in common. With infinite streams we face a significantly more complex situation than with finite sequences (as seen in (1)). As an example we use a set of transactions $\mathbb{N}$ and as state use the set of all transactions which the participant has seen so far. Thus $s_0 = \emptyset$. One possible development of the sequences for two particular streams is illustrated in Fig. 1 and shows the phenomenon.

Byzantine Failures: Participants misbehave, by bad intent or by malfunction. They send different transactions to some participants, tricking them into inconsistent states.

[5] Byzantine failure mode assumes that a node may exhibit completely arbitrary, random, even malicious behavior.

[6] Bitcoin is intended as a non-stop running system – there is no final state to be reached.

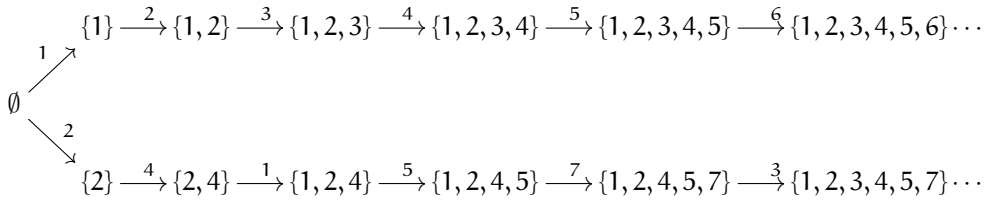

Figure 1: Two sequences of local states, which share only the initial local state.

Network Partitioning may happen without the participants noticing. This produces several connected components, each of which is unaware of the transactions taking place in the other components.

2.3 Distributed Applications and Swarm Behavior

Applications impose consistency criteria on the states of the participants. If ALICE wants to pay 1$ to BOB, both parties must have the same view on the balances before and after the transaction, reflecting that 1$ is moved between the accounts. MALLORY might attempt fraud and engage ALICE and BOB in simultaneous transactions, trying to use the same 1$ to pay both parties. We must protect the users from such fraud: Some time after the transaction is completed, the new owner(s) of the transferred coin(s) must be clear and the sum of money in circulation must have stayed the same. For other applications, the requirements are accordingly. Usually, centralized solutions implement consistency by well-known transactional concepts, distributed architectures use serialization protocols.

Bitcoin takes a fresh approach. It rejects the idea of trust in a single entity. It assumes a (very) large number of participants ($\sim 10^6$), which kills traditional serialization approaches. Furthermore, Bitcoin not even requires a common code base: Every node exchanges messages with other nodes, and it is by a probabilistic game theoretic incentive that the swarm *seems to pursue* a common goal, which, hopefully, attracts a sufficient number of cooperative nodes, which only in turn guarantee the incentive, which was the reason to participate in the first run. In this sense, Bitcoin is a self-organized, emergent and (hopefully) self-stabilizing network of communicating nodes.

In the Bitcoin community, a more traditional, governed understanding is prevailing: A network of peer-to-peer nodes, running in their vast majority a recent version of the official Bitcoin software. A few nodes may deviate from the "official" behavior, which is tolerated by the probabilistic consensus as long as only a minority engages in counter-specification behavior. In Bitcoin, minority does not refer to numbers of participants but computing power, as demonstrated by a proof-of-work mechanism.

This more traditional interpretation may be due to social, economic and psychological effects: The active member of the community invests time, money (for running nodes) and entrepreneurial expectations. This is easier, if it is based on a belief in a stable system, well-governed by a solid architecture and endangered only by a small minority of malign attackers. This belief might be an appropriate (and stabilizing) social model; from a struc-

tural point of view, however, Bitcoin is an anarchic network of interacting but completely autonomous nodes, whose seemingly goal-directed behavior emerges from advantages the participants expect from their predictions on the average behavior of a majority of nodes.

Network partitioning is dealt with by assuming that in contemporary Internet a long-term partitioning into disconnected components will not go unnoticed by the participants and by hiding conflict resolution upon rejoining in the resilience properties against malicious minorities. **Byzantine failures** and **unreliable communication** are dealt with in a statistical manner; the algorithm aims that states of nodes eventually converge to a consistent world view.

3 Transaction Networks

For further formalization, we find the notion of a transaction network helpful. It replaces consensus on state by *eventual consensus* on *past history*, and is able to accommodate a *"more" distributed perspective* than a linear sequence of states. An extended version of this formalism is suitable for modeling more general distributed and parallel processes, see [Cap00].

3.1 Definition

A transaction network (tx-net) $(\mathcal{Q}, \mathcal{T}, \rightarrow)$ consists of finite sets $\mathcal{Q}$ of *anchors*, $\mathcal{T}$ of transactions and a relation[7] $\rightarrow \subseteq \mathcal{Q}^* \times \mathcal{T} \times \mathcal{Q}^*$.

In $q_1 q_2 q_3 \xrightarrow{t} p_1 p_2$ the anchors $q_\cdot$ of the left side are *inputs* and the anchors $p_\cdot$ on the right side are *outputs* of t. We require that for every transaction t there is exactly one input word $q \in \mathcal{Q}^*$ and one output word $p \in \mathcal{Q}^*$: $\forall t \in \mathcal{T} : \exists!(q,p) \in \mathcal{Q}^* \times \mathcal{Q}^* : q \xrightarrow{t} p$. We define the set $\alpha(t) := \{q_1, \ldots, q_a\} \subseteq \mathcal{Q}$ of input-anchors and the set $\omega(t) := \{p_1, \ldots, p_b\} \subseteq \mathcal{Q}$ of output-anchors of t. A transaction with empty input word ε has $\alpha(t) := \emptyset$ and is called a *generating* transaction[8], a transaction with empty output word has $\omega(t) := \emptyset$ and is called a *deleting* transaction[9]. We may[10] require that every anchor $q \in \mathcal{Q}$ must occur in the output of exactly one transaction (which is the transaction generating this anchor). Moreover every anchor may occur in the input of at most one transaction (which is the transaction consuming this anchor). An anchor which does not occur in the input of any transaction is *unspent*, otherwise it is *spent*. Finally, we require a tx-net to be *cycle-free*: There is no k-tuple $(t_1, t_2, \ldots, t_k)$ of transactions, $k \geq 1$, such that $\omega(t_1) \cap \alpha(t_2) \neq \emptyset$, $\omega(t_2) \cap \alpha(t_3) \neq \emptyset$, $\ldots$ $\omega(t_k) \cap \alpha(t_1) \neq \emptyset$, transactions do not feed into themselves, even after transitive closure. We may visualize tx-nets as in Fig. 4, illustrating transactions a, b, c by boxes and anchors x, y, z by circles.

We define an immediate successor relation $\triangleright \subseteq \mathcal{T} \times \mathcal{T}$ by $t_1 \triangleright t_2 \;\; :\Leftrightarrow \;\; \omega(t_1) \cap \alpha(t_2) \neq \emptyset$, which means that transaction t_1 produces at least one anchor as output which serves as

[7] As usual, $\mathcal{Q}^* := \cup_{n=0}^{\infty} \mathcal{Q}^n$ and $\mathcal{Q}^+ := \cup_{n=1}^{\infty} \mathcal{Q}^n$ with $\mathcal{Q}^0 =: \{\varepsilon\}$ and $\varepsilon \notin \mathcal{Q}$.

[8] In Bitcoin terminology, this is called a coinbase transaction.

[9] Deleting transactions do not occur in the economic application of Bitcoin but could be useful in other scenarios.

[10] Equivalently, we may designate certain states as initial states; the difference is purely technical.

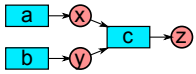

Figure 2: Single transaction network.

input to t_2 (t_1 feeds into t_2). $\rightarrow$ is cycle-free, the reflexive-transitive closure $\triangleright^*$ of $\triangleright$ is a (partial) order $\leq := \triangleright^*$ on $\mathcal{T}$. $t_1 \leq t_2$ indicates that t_1 has *occurred earlier* than t_2.

A tx-net describes the *evolution of state*, i. e. how an initially empty state of a participant evolves into the current state. its more abstract view removes the superfluous details encoded in a linear sequence of states and focuses on consensus beyond coincidental ordering. In Fig. 2, the tx-net abstracts away from the inessential fact that the current state z could have been reached via the linear transaction sequence abc or via bac; it focuses on the fact that it has been reached by executing one of several possible orderings of transactions. A tx-net, however, still distinguishes more histories than necessary for a state-based consensus. The tx-nets in Fig. 3 both produce the same state and thus allow for state-based consensus; however they do not allow for history-based consensus. The same is true for Bitcoin, where a different transaction history will always produce a conflict in the block chain, although there might be consensus on the values of the various accounts.

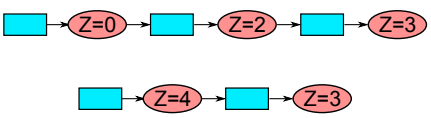

Figure 3: Upper and lower tx-nets produce the same current state.

To describe the *evolution of tx-nets* we define a (partial) order on the set of all tx-nets $\mathfrak{N}$. A tx-net N_1 evolves into another tx-net N_2, if N_2 is obtained by attaching a single transaction to N_1. This immediate successor relation $\rightsquigarrow \subseteq \mathfrak{N} \times \mathfrak{N}$ on the set of all tx-nets produces a (partial) order $\preceq := \rightsquigarrow^*$ by taking the reflexive, transitive closure as above. If N_1 evolves into N_2 (i.e. $N_1 \rightsquigarrow N_2$) then tx-net N_1 will also be a smaller network than N_2 (i.e. $N_1 \preceq N_2$). Fig. 4 illustrates the immediate successor relation on four tx-nets. Moreover, the figure shows how the centralized scenario (consisting of states r, s, t, u, transactions a, b, c, d and $\nu(a, r) = s, \nu(b, s) = t, \nu(c, r) = u, \nu(d, u) = t$ and $\dagger$ for all other combinations) translates into a distributed scenario.

3.2 Convergence of Consensus

For two tx-nets N_1, N_2 we define the intersection or common core $N_1 \sqcap N_2$ as the largest[11] tx-net which may be extended into N_1 and N_2 (i.e. $N_1 \sqcap N_2 \preceq N_1$ and $N_1 \sqcap N_2 \preceq N_2$).

[11] It is not immediately evident that such an element exists, but can be shown by induction or by using results from section 3.6 of [Cap00].

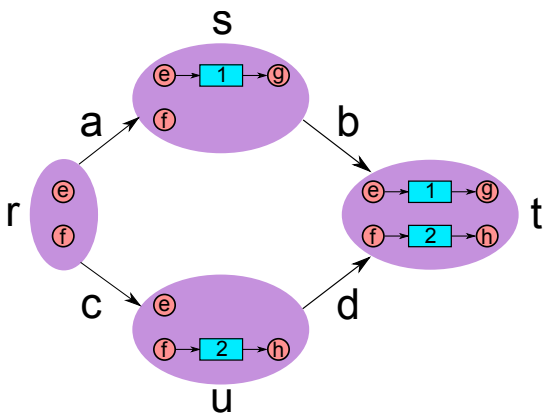

Figure 4: Successor relation on transaction networks.

We assume a situation where two participants P, Q both start with the empty tx-net and subsequently build "towards" the same tx-net, however in a manner where the order of adding transactions may differ. Thus P will produce the sequence $\varepsilon = p_0 \rightsquigarrow p_1 \rightsquigarrow p_2 \rightsquigarrow \ldots$ of tx-nets and Q will produce the sequence $\varepsilon = q_0 \rightsquigarrow q_1 \rightsquigarrow q_2 \rightsquigarrow \ldots$

It may be the case that for every q_n there exists a p_m such that $q_n \preceq p_m$; this means that every state change (i. e. transaction) seen at Q will eventually be seen by P. If this holds vice versa, then both participants follow the same sequence of transactions, just sometimes one of the participants will know a bit different transactions than the other due to different sequencing, but eventually both will catch up. The states on which both participants will agree is given by the sequence of the common core, i. e. $p_0 \sqcap q_0, p_1 \sqcap q_1, p_2 \sqcap q_2, \ldots$

To guarantee such a common limit for all participants is the task of the Bitcoin algorithm: After a while all participants should have agreed on a common past (and on the produced states); the moment might be earlier or later, depending on the communication latency.

We conjecture that this concept of a common limit may be substantiated with domain-theoretic methods, where the order structures and infinite limits necessary for this approach are already in place. We doubt, however, that such a description will be very useful for a practical understanding of the algorithm, which at every finite time step will always have produced *finite* tx-nets, whereas the approximated limit objects will have to be an infinite variant of tx-nets and thus will never be seen on a real computer. The tools for describing convergence, nevertheless may be helpful from a theoretical point of view and could be helpful for proving properties of the algorithm. This speculation, however, has to be substantiated in future work.

3.3 Backtracking

Unfortunately some problems remain. Fig. 5 shows a tx-net r which may develop into tx-net s or into an *incompatible* tx-net u: It is no longer possible that these two tx-nets

become parts of a common larger tx-net and thus regain consensus at a later stage! This is fundamentally different from Fig. 4, where $\mathtt{r}$ splits into $\mathtt{s}$ or $\mathtt{u}$, but a later consensus is possible in the form of tx-net $\mathtt{t}$.

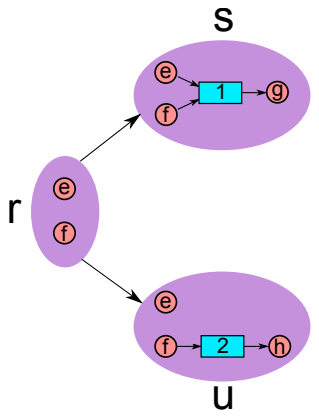

Figure 5: Split in tx-net evolution.

Such a situation may arise when a node M sends a transaction $\mathtt{t}_1$ to one set $Q_1 \subseteq Q$ of participants and a carefully chosen, different transaction $\mathtt{t}_2$ to another, disjoint set $Q_2 \subseteq Q$ of participants, where $\mathtt{t}_1$ and $\mathtt{t}_2$ are chosen as[12] in Fig. 5 and not as in Fig. 4. In Bitcoin, a double spending attack is an example of such a situation.

Similarly, a node $\mathtt{p}_1$ might broadcast a transaction $\mathtt{t}_1$ and at nearly the same time node $\mathtt{p}_2$ might broadcast a transaction $\mathtt{t}_2$ leading to the described situation. This might be a result of a malicious collusion between the two nodes, it could occur due to an (unintended) application-level coordination failure of these nodes, or it could be part of (intended) application behavior, where the application assumes that the problem will be fixed by the block chain mechanism.

In all these cases, the desired convergence to a consistent view of the past is disrupted and some nodes must undo parts of their transaction history. The goal is not to construct the "correct" state of transaction history (we have no criterion what "correct" means), but to ensure eventual convergence to one consistent transaction history. In Bitcoin this is achieved by the block chain.

3.4 Regaining Consensus

In the interest of consensus the nodes exchange messages informing each other of what they believe how the consensus should look like. For this purpose a node sends its entire sequence of tx-nets to its peers[13]. From this information the receiving node can construct

[12] The intuitive meaning should be clear from the figures. A mathematically precise definition is straight forward but needs additional formalism which we skip for lack of space.

[13] In Bitcoin this is implemented in a more efficient way using hashes and inventory control messages, whose implementational details obscure the concepts.

a cycle-free graph of tx-nets, which includes its own tx-net sequence corresponding to its local state. Fig. 6 shows an example of such an evolution structure. Moreover, Fig. 6 also demonstrates why it is not enough when the other nodes only send transactions: It is not obvious, whether a transaction attaching to $z = 2$ should be added to the upper or to the lower evolution path of tx-nets; only if the nodes send entire sequences of tx-nets, this question may be settled.

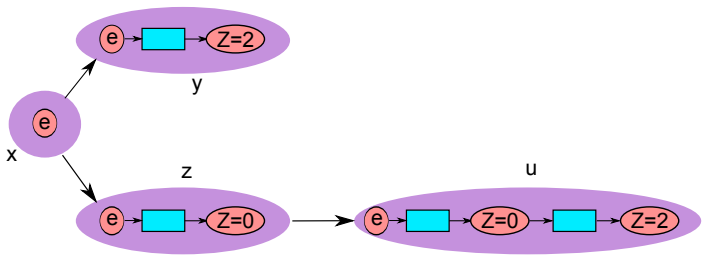

Figure 6: Evolution structure of tx-nets.

A node thus has some[14] information about how the local states of the other nodes are evolving.

Every moment a node will have a notion which tx-net it considers to be "true". To this end it will chose a maximal element with regard to the $\preceq$ order on the tx-nets in its evolution structure. Over time, a node will thus produce a sequence of such maximal elements. Backtracking means that this sequence $\varepsilon, \mathfrak{m}_1, \mathfrak{m}_2, \mathfrak{m}_3, \ldots$ will not be linearly ordered $\varepsilon \preceq \mathfrak{m}_1 \preceq \mathfrak{m}_2 \preceq \mathfrak{m}_3 \ldots$ but may contain incomparable elements.

We again refer to Fig. 6: A node might start with an evolution structure consisting only of $\mathfrak{x}$, selecting $\mathfrak{x}$ to be its "true" tx-net. Due to communication, it may learn of the tx-nets $\mathfrak{z}$, $\mathfrak{y}$ and $\mathfrak{u}$ (in this sequence). The node might then first decide to adopt $\mathfrak{z}$ as its subsequent truth, then it might switch over to $\mathfrak{y}$ and later back again to $\mathfrak{u}$.

In the current Bitcoin implementation, the block chain is used to implement the evolution structure. Every block corresponds to a tx-net. The block chain realizes a tree. This is no additional limitation, since the unique addresses of the coinbase transactions will always prevent the evolution structure from having additional, non-tree order relations; this is specific for Bitcoin and need not be the case in different applications of the same consensus mechanism. Moreover, Bitcoin always chooses the block with the highest proof-of-work as the "true" tx-net. This is specific for the current Bitcoin implementation; there are other approaches, which base this decision on a *proof-of-stake* maximum (see below).

[14]The information may be incomplete, since a node does not communicate with all the other nodes; due to communication latency, it may not reflect the latest changes; moreover, some aspects might be lost, since communication is not reliable.

4 Swarm Behavior

Bitcoin nodes receive, send and forward messages; in the mean time, they process information (the most common task is checking received blocks and solving proof-of-work puzzles). A node does so due to one of the following reasons:

(1) **Incentive:** The node expects benefits from interacting with other nodes and attempts to maximize this benefit.

(2) **External motivation:** The node disregards transactional benefits and pursues an external motivation. For example, a participant might be interested in disrupting the network.

(3) **Failure:** The node does not follow a predictable pattern but acts erroneously.

What is a good mathematical model for describing the behavior emerging from a decentralized swarm of communicating (Bitcoin) nodes?

4.1 Bitcoin as a Game

The incentive mechanism suggests modeling Bitcoin by game theory [Web06]: A certain (large) number of participants is playing a repeated number of rounds. In every round, a participant may chose between a number of options, which in Bitcoin correspond to sending messages of a specific content to other participants. Since Bitcoin is not operating in synchronized rounds and with a fixed number of nodes, we include the option of not sending a message. The random nature of a proof-of-work may be reflected by mixed strategies and by placing constraints on the respective probabilities. From this perspective, a node decides between sending a correctly found new block (with a probability constrained by its hash performance), not sending a message, or sending an incorrect block. According to game theory, the goal of a node is to select its behavior in such a manner as to maximize the payoff in the game.

Unfortunately, there are serious problems with this approach. In repeated games, the payoff often is obtained by summing the payoffs of all rounds. However, in Bitcoin a payoff is not connected with a single round, but evolves over time and retains a stochastic nature.

For example: An attacker might double spend to two different Bitcoin exchanges and subsequently ask them to convert his Bitcoins into Dollars. There is a certain probability that the attack succeeds and both exchanges will pay the appropriate amount in Dollar before realizing the attack by a block chain reorganization. This probability depends on the confirmation policy of the exchanges[15]. Thus, the payoff depends on a large number of events, corresponding to many rounds in the game. Even worse: The probability that a Bitcoin transaction will be undone and the block chain will reorganize never becomes zero. The payoff becomes certain only when a participant finally crosses the boundary from digital (Bitcoin) to real (Dollar) economy. It is generally *believed* that this probability is so small that the impact can be neglected for practical purposes. However, when in search of a modeling tool which is able to rigorously *prove* this property we must not base the choice of the tool on the assumption we want to prove.

[15] i. e. how many blocks they wait until they do a bank transfer.

4.2 Bitcoin as Random Walk

A more suitable tool might be a random walk, where the length of the walk provides a measure for the probability of reverting a transaction. This is the method chosen by SATOSHI in the Bitcoin white paper [Nak09]. The reasoning employed is very simplified, considers only two players and neglects various kinds of effects such as communication latency, nodes joining and leaving the network, or evolutionary changes in the behavior of the nodes. It provides a persuasive reason but not a stringent proof. As soon as other aspects are taken into account, the resulting stochastic process seems to become too complex to obtain hard results.

Our main critique with this approach is as follows: The approach *assumes* that a majority of nodes will adhere to the "official" Bitcoin protocol. It then argues in [Nak09] that a minority[16] of nodes, which deviate from the official protocol, will not be able to damage the incentive, which is produced by the "official" protocol. This observation is important, since it guarantees Bitcoin stability of a convinced majority against a malevolent minority. For a completely decentralized system, this is not enough! There is no authority ensuring a majority or even a notion of an "official" protocol version. Every participant is free to switch to a modified protocol; he will switch, if this increases his payoff. The *assumption*, that a majority of nodes will stick to the "official" protocol is not accounted for or, in other versions of the debate, is explained (using *circular reasoning*) by refering to the incentive generated by the "official" protocol. The reasoning does not explain, why a majority of nodes would *want* to stick to the "official" or any other currently used version of the protocol. The inherent dynamics of the protocol with regard to bounty-size and difficulty, as well as the growing interest by the open source community, is likely to further drive this issue by suggestions of new protocol variants. This discussion is important and at the core of the future stability of the Bitcoin system.

4.3 Bitcoin as Social System

Both theories, game theory and random walks, assume a behavior where nodes attempt to maximize their profit. A thought experiment may expose that this is not the only driving force present in Bitcoin behavior.

Currently Bitcoin uses a bounty of 50 BTC for new blocks. Suppose, a node decides to change this parameter to 500 BTC. In a maximum profit approach, all participants, at least those operating in incentive mode, will adopt their behavior to maintain optimal profit. An appropriate reaction for the other nodes would be to adopt this parameter change and continue the block chain with modified constraints: For all mining nodes, the mining yield would rise; the non-mining nodes would not see a difference.

Of course, there are numerous reasons why we do not see such a parameter change in real-life Bitcoin networks: Higher mining yields might cause inflation and compensate the increased Bitcoin payoff by a dip of the Bitcoin to Dollar conversion rate. Adoption of a new parameter is not an immediate rational decision but requires the participants to reach some form of social consensus. Finally, current Bitcoin software has the bounty encoded

[16] weighted by their hash computation performance

into the program – it cannot be changed easily. Thus, social effects and human inertia currently play an important role in Bitcoin stability.

However, in principle parameters may be changed much faster: Users could be offered option dialogues or optimization algorithms could choose them automatically. In such a situation the current operative stability of Bitcoin could be less obvious. In stock trading, algorithmically induced course artifacts and crashes are a well-known and validated phenomenon [JZH+12].

We therefore *conjecture* that a proper model of Bitcoin needs to include behavioral elements beyond rational profit maximization, a belief which is supported by results in human decision making [GS02]. Conceptually, society[17] introduce an element of centralized trust into an otherwise completely decentralized architecture.

This leads to several interesting questions: How can these additional elements be modeled in a rigid mathematical manner? How does swarm behavior emerge from interacting nodes? Is an incentive sufficient? Does the incentive emerge in the swarm – or is an external incentive (eg. convertibility into dollars) necessary? Is the behavior of the swarm stable if the individual players are completely free to adapt their behavior in order to maximize received incentives?

These questions are far from being purely philosophical since their answers are the tools required to settle security questions currently debated in the Bitcoin community: *Heard-of Byzantine consensus* lets the swarm chose "correct" behavior by a one-identity-one-vote scheme and is susceptible to sybil attacks[18]. *Proof-of-work* improves this by counting the votes according to node hash performance. *Proof-of-stake*[19] counts the votes according to the stake a node currently has in the Bitcoin system. The latter may be defined by the bounty acquired through past mining activities or by the total wealth currently held. When social aspects are a core aspect in Bitcoin stability, stake measured in real economic units might prove more important that hash performance.

We *conjecture* that a suitable modeling discipline will contain game theoretic and evolutionary elements; in contrast to the established discipline of evolutionary game theory it should not limit evolution to a participant's choice of strategies or to specific game parameters but would allow for wider modifications of the set of available game strategies and connected payoff mechanisms. Some starting points might be found in [SF07].

5 Related Questions

It would be interesting to analyze Bitcoin from an *algorithmic perspective* in addition to a structural one: Bitcoin attempts to solve a Byzantine consensus problem, which has been well studied for decades.

Deterministic Byzantine agreement needs at least as many rounds of communication as the number of faulty processors [BO83]. This is a show-stopper for all deterministic ap-

[17] For example, elements of consensus of the Bitcoin software developers when roling out new software.

[18] In a sybil attack multiple (fake) identities are generated by a single user and thus can bias the vote.

[19] See https://en.bitcoin.it/wiki/Proof_of_Stake for the algorithm and its implementation; see https://bitcointalk.org/index.php?topic=37194.msg456773 for the debate.

proaches: Sybil attacks can simulate a large number of faulty processors and drive the communication complexity. The probabilistic proof-of-work in Bitcoin allows an escape from the deterministic complexity bounds and requires contributing nodes to make substantial investments in hash performance.

There is considerable state of the art on *probabilistic consensus*. [BO83] describes an algorithm which is correct with probability 1 in the long run, tolerates less than $n/5$ faulty processors[20] but may require an exponential number of rounds; if the number of faulty processors is $\mathcal{O}(\sqrt{n})$, consensus can be reached in a protocol where the expected number of rounds is independent from n.

Many authors build upon this approach and focus on improving complexity bounds [Bra87], [ADH08], [Zam96], with [KS10] and [KKK+10] arguing that all essential complexity bounds can be made *polylogarithmic*.

Other authors concentrate on specific algorithmic aspects, such as the number of tolerated faulty processors [CVL10], buffer requirements [BEV06], synchronicity requirements [DDS87], or varying numbers of participants in ad-hoc networks [ADGF08].

While Byzantine consensus is generally accepted as very important problem and numerous algorithms exist, only a very small number of solutions have been rigorously or formally verified; this is especially true for algorithms which do not use the *heard-of* model of building consensus [CBDM11]. This is, unfortunately, also true of the Bitcoin idea.

All non-Bitcoin approaches to the agreement problem known to the author require limits on the *number* of faulty processors. Therefore, they are an easy victim to Sybil attacks, which simulate large numbers of faulty nodes. Sybil attacks in general may be prevented by trust concepts or by singling techniques. *Trust concepts* connect a digital identity with a real identity. They may be implemented by a *trusted certificate authority* or by *identity-based authentication* [BNSnS04], both of which often use a single trusted third party. They are in conflict with Bitcoin's requirement of not relying on a single, centralized element. They may be built on *distributed mechanisms* [ARH97], which need preparatory steps in a network of mutually trusting users and are less suitable for ad-hoc scenarios. *Singling techniques* guarantee that a person owns only one recognized handle and may be implemented, for example, by pseudonym parties [FS08], which again require preparatory steps and infrastructure, which is not yet existing. Sybil attacks and decentralized trust are very active research areas; but most approaches are in conflict with Bitcoin requirements (eg. no trusted third party, no central registry, some degree of anonymity), are essentially equivalent to solving a consensus problem (and thus equivalent to the Bitcoin problem itself), need additional infrastructure or have not yet left their research state.

Acknowledgement

Special thanks of the author go to the anonymous referees, especially to one referee who provided a substantial list of suggestions, including typesetting remarks, all of which helped to improve the quality of the paper.

[20] As usual, n denotes the number of all nodes.

References

[ADGF08] Mohssen Abboud, Carole Delporte-Gallet, and Hugues Fauconnier. Agreement and consistency without knowing the number of processes. In *Proceedings of the 8th international conference on new technologies in distributed systems*, NOTERE '08, pages 38:1–38:8, New York, NY, USA, 2008. ACM.

[ADH08] Ittai Abraham, Danny Dolev, and Joseph Y. Halpern. An almost-surely terminating polynomial protocol for asynchronous byzantine agreement with optimal resilience. In *Proceedings of the twenty-seventh ACM symposium on principles of distributed computing*, PODC '08, pages 405–414, New York, NY, USA, 2008. ACM.

[ARH97] Alfarez Abdul-Rahman and Stephen Hailes. A distributed trust model. In *Proceedings of the 1997 workshop on new security paradigms*, NSPW '97, pages 48–60, New York, NY, USA, 1997. ACM.

[BEV06] François Bonnet, Paul Ezhilchelvan, and Einar Vollset. Quiescent consensus in mobile ad-hoc networks using eventually storage-free broadcasts. In *Proceedings of the 2006 ACM symposium on applied computing*, SAC '06, pages 670–674, New York, NY, USA, 2006. ACM.

[BNSnS04] Joonsang Baek, Jan Newmarch, Reihaneh Safavi-naini, and Willy Susilo. A Survey of Identity-Based Cryptography. In *Proc. of Australian Unix Users Group annual conference*, pages 95–102, 2004.

[BO83] Michael Ben-Or. Another advantage of free choice (Extended Abstract): Completely asynchronous agreement protocols. In *Proceedings of the second annual ACM symposium on principles of distributed computing*, PODC '83, pages 27–30, New York, NY, USA, 1983. ACM.

[Bra87] Gabriel Bracha. An O(log n) expected rounds randomized byzantine generals protocol. *J. ACM*, 34(4):910–920, October 1987.

[Cap00] Clemens Heinrich Cap. *A Calculus of Distributed and Parallel Processes*. Teubner, 2000.

[CBDM11] Bernadette Charron-Bost, Henri Debrat, and Stephan Merz. Formal verification of consensus algorithms tolerating malicious faults. In *Proceedings of the 13th international conference on stabilization, safety, and security of distributed systems*, SSS'11, pages 120–134, Berlin, Heidelberg, 2011. Springer-Verlag.

[CVL10] Miguel Correia, Giuliana S. Veronese, and Lau Cheuk Lung. Asynchronous Byzantine consensus with 2f+1 processes. In *Proceedings of the 2010 ACM symposium on applied computing*, SAC '10, pages 475–480, New York, NY, USA, 2010. ACM.

[DDS87] Danny Dolev, Cynthia Dwork, and Larry Stockmeyer. On the minimal synchronism needed for distributed consensus. *J. ACM*, 34(1):77–97, January 1987.

[DSU04] Xavier Défago, André Schiper, and Péter Urbán. Total order broadcast and multicast algorithms: Taxonomy and survey. *ACM Comput. Surv.*, 36(4):372–421, December 2004.

[EFLF07] Patrick Eugster, Pascal Felber, and Fabrice Le Fessant. The "art" of programming gossip-based systems. *SIGOPS Oper. Syst. Rev.*, 41(5):37–42, October 2007.

[FS08] Bryan Ford and Jacob Strauss. An offline foundation for online accountable pseudonyms. In *Proc. of the 1st international workshop on social network systems SocialNets*, 2008.

[GS02] Gerd Gigerenzer and Reinhard Selten. *Bounded Rationality: The Adaptive Toolbox*. MIT Press, 2002.

[JZH+12] Neil Johnson, Guannan Zhao, Eric Hunsader, Jing Meng, Amith Ravinadr, Spencer Carran, and Brian Tivnan. Financial black swans driven by ultrafast machine ecology. arXiv:1202.1448v1 [physics.soc-ph], February 2012.

[Kam11] Dan Kaminsky. Black Ops of TCP/IP 2011. Black Hat USA, http://dankaminsky.com/2011/08/05/bo2k11/, 2011.

[KKK+10] Bruce M. Kapron, David Kempe, Valerie King, Jared Saia, and Vishal Sanwalani. Fast asynchronous Byzantine agreement and leader election with full information. *ACM Trans. Algorithms*, 6(4):68:1–68:28, September 2010.

[KS10] Valerie King and Jared Saia. Scalable byzantine computation. *SIGACT News*, 41(3):89–104, September 2010.

[Nak09] Satoshi Nakamoto. Bitcoin: A Peer-toPeer Electronic Cash System. Self-published white paper at http://bitcoin.org/bitcoin.pdf, 2009, 2009.

[RJ08] Benjamin Reed and Flavio P. Junqueira. A simple totally ordered broadcast protocol. In *Proceedings of the 2nd workshop on large-scale distributed systems and middleware*, LADIS '08, pages 2:1–2:6, New York, NY, USA, 2008. ACM.

[SF07] György Szabó and Gábor Fáth. Evolutionary Games on Graphs. *Elsevier Physics Reports*, 446:97–216, 2007.

[Web06] James N. Webb. *Game Theory: Decisions, Interaction and Evolution*. Springer, 2006.

[Wir90] Martin Wirsing. Algebraic Specification. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science (vol. B)*, pages 675–788, Cambridge, MA, USA, 1990. MIT Press.

[Zam96] Arkady Zamsky. An randomized Byzantine agreement protocol with constant expected time and guaranteed termination in optimal (deterministic) time. In *Proceedings of the fifteenth annual ACM symposium on principles of distributed computing*, PODC '96, pages 201–208, New York, NY, USA, 1996. ACM.