Blockchain-Free Cryptocurrencies:
A Framework for Truly Decentralised Fast Transactions

Xavier Boyen — QUT
Christopher Carr — NTNU & QUT
Thomas Haines — QUT

Abstract. The "blockchain" distributed ledger pioneered by Bitcoin is effective at preventing double-spending, but inherently attracts (1) "user cartels" and (2) incompressible delays, as a result of linear verification and a winner-takes-all incentive lottery.

We propose to forgo the "blocks" and "chain" entirely, and build a truly distributed ledger system based on a lean graph of cross-verifying transactions, which now become the main and only objects in the system. A fully distributed consensus mechanism, based on progressive proofs of work with predictable incentives, ensures rapid convergence even across a large network of unequal participants, who all get rewards working at their own pace. Graph-based affirmation fosters snappy response through automatic scaling, while application-agnostic design supports all modern cryptocurrency features such as multiple denominations, swaps, securitisation, scripting, smart contracts, etc.

We prove theoretically, and experimentally verify, our proposal to show it achieves a crucial "convergence" property — meaning that any valid transaction entering the system will quickly become enshrined into the ancestry upon which *all* future transactions will rest.

1 Introduction

Before Bitcoin, earlier cryptographic digital cash schemes relied on trusted authorities [8]. Bitcoin reformed this classical view of a cryptocurrency, trading centralised clearing for a distributed ledger, secured by the majority rule without pre-ordained authority. Quite remarkably, it did so using only standard cryptographic primitives such as signatures and hash functions.

Although other cryptocurrencies such as Bit-gold [4] and B-money [1] have been credited [28] with anteriority amongst authority-free cryptocurrencies, Bitcoin is the first to see widespread adoption. Various factors may account for Bitcoin's success, though there is little question that the decentralised design and perceived immunity to government interference played a significant part in its uptake. There is now a surge of academic and layman interest in distributed crypto-cash, and, more specifically in "Blockchain technology", hailed as one key innovation of Bitcoin (the other being transaction scripting, paving the way for "smart contracts" [20]).

Alas, the lack of any *imposed* authority did not prevent the emergence of oligopolies, or "mining pools", which would have consolidated into an unshakable monopoly if not for ostensible self-restraint to maintain user confidence within the system. Moreover, with the influx of new users, and older users seeking to consolidate their grip on the system, it did not take long for problems to emerge, leading to arguments questioning the fundamental design and pressing for changes.

Some problems unfortunately cannot be fixed without a complete redesign [2, 6, 14, 21, 23, 24, 36]. The range of proposals goes from mere tweaks, or *altcoins* [10, 22], to infrastructure redesigns, often with in-built centralisation [32] or even sovereign escrow [7] as "solutions" to the perceived problems. Less controversial proposals focused on better anonymity [3, 24] and multi-denomination smart contracts [12, 35].

The bigger issues that remain include the ineluctable consolidation of coin "min(t)ing" into *mining pools*, and incompressibly long verification delays. We argue that both problems are *inherent* to blockchain technology itself, as a side-effect of it not being truly decentralised.

Mining Pools and Centralization. The lack of scalability observed within Bitcoin is an artefact of the Blockchain consolidation principle itself, which makes it very hard to distribute rewards to the myriad of participants that contribute (or would be contributing) their computing power to the verification effort. On the linear Blockchain, the rewards are few and far between, and the only fair and secure method of distributing them is in essence a lottery.

Risk-averse participants coalesce into mining pools to reduce the variance, at the cost of abdicating their individual oversight duty. This collectively makes the network more brittle to a variety of attacks, not only the well-known *51% attack* [27], but also the "selfish miner" or *33% attack* [13], which in some special cases can become a *25% attack*. Proposals have been made to mitigate this issue, by incentivizing participants to defect from pools [26], but they do not eliminate the Blockchain's core problem of low-odds high-variance rewards.

Mining pools come with the fundamental problem that they are themselves becoming monopolies within the system, defeating the principle of decentralisation. As Bitcoin developed, mining pools took a more and more significant position. Mining pools now make up more than 99% of the (considerable) *hashpower* in the Bitcoin system. A single one of these monoliths temporarily held absolute majority of the computing power on the network [16]. At the time of writing, those mining pools are mostly based in a single country (China), with just *four* of these pools controlling over 50% of the hashpower of the network.[1]

[1]Accurate per `blockchain.info` at 10 February 2017.

This move away from decentralisation is a widely recognised problem, and is one that has come under scrutiny in the past [21, 26]. Tackling this issue has seen some lines of inquiry seek to give more power to central authorities in an attempt to address power wastage and scalability [9].

Inherent Verification Waiting Times. Bitcoin uses a feedback-loop mechanism on the puzzle solving time, altering the difficulty to ensure that new blocks are created every 10 minutes on expectation. Unfortunately, 10-minute expected delays are troublesome, especially coming from a stateless, high-variance, Poisson process. For miners, this is barely enough to ensure that they quickly learn of new blocks to avoid wasting work (and perversely encourages them to delay the propagation of new blocks other than their own). For users, a long and unpredictable verification delay hinders Bitcoin for any significant real-time payments. Ironically, commercial Bitcoin users now mitigate the blockchain delays by entrusting third parties as "payment processors", the very thing that Bitcoin sought to avoid.

Further compounding these inherent delays, Bitcoin presently caps the block size, which creates contention within the community and may increase delays: a problem that miners have no incentive to address. Incidentally, block sizes have been raised once already, from 4KB to 1MB, and is presently the object of a *fork*, with heated debate within the community.

In short, contention and consolidation are inherent to the Blockchain reward structure, making it unsuitable to cater for a large population of miners and users.

1.1 Our Approach

Our Blockchain-free proposal shifts onto the transactions themselves the task of affirming prior transactions. Verification no longer results in a chain of transactions blocks, but in a lean graph comprised only of transactions.

A Graph of Cross-Verifying Transactions. When a transaction is posted, in addition to any actual "cryptocurrency" payload, it will refer to two previous transactions deemed to be valid, bundled with some proof of work. This induces a growing "hash-graph" of verifications, where each transaction verifies two "parents". Growth of the graph is steered using a very fine-grained incentive system that encourages affirming recent transactions while equitably rewarding all transactions, even when they share the same parents (up to some limit). Implicit rather than explicit collaboration is rewarded, encouraging users to work alone instead of forming pools, while reducing the waste of verification effort.

Figure 1 illustrates how transactions refer to each other. The arrows point from a *parent* to a *child* transaction. Figure 2 shows the same image,

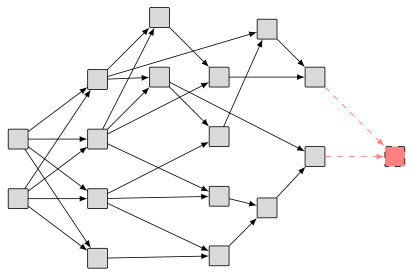

Figure 1: Transactions Graph.

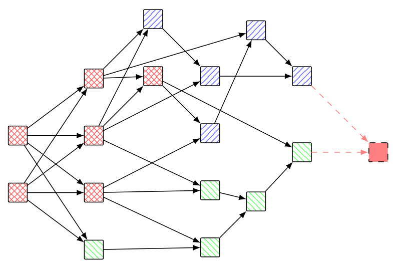

Figure 2: Counting Ancestors.

highlighting the ancestor transactions of the new (dashed arrow line) transaction on the far right. The crosshatched nodes (in red), represented transactions that are common ancestors of both the right and left parent nodes. The diagonally shaded (blue and green) transactions, represent transactions from one parent but not both. Eventually every valid transaction ends up being verified, directly or indirectly, by *all* new valid transactions, making the transactions as immutable as in a linear block-chain. We refer to this as *convergence*, and say that a set of transactions have converged if they all share a common descendant in the graph.

The crucial part of the framework is the realisation of the security and consensus mechanisms. In order to achieve a distributed consensus, it is necessary to calculate the combined proof-of-work contributed on each transaction by itself and its descendant transactions. In order to accurately combine proofs-of-work, one must be able to count descendant and any ancestor transactions without double-counting overlapping contributions of computational effort. This process is necessary for efficient verification.

In Section 3, we prove that convergence does occur, and further in Appendix A, we experimentally verify that convergence occurs rapidly, which is essential for securing and scaling up the system. To our knowledge, this work is the first to propose an entirely block-free graph-based cryptographic ledger which is robust against attacks and suitable for cryptocurrencies.

Proofs-of-Work (PoW). The proof-of-work concept was initially developed by Dwork and Naor [11] to fight spam. Other applications include client puzzles [18, 34], but the most famous one to date remains the Bitcoin blockchain itself [27].

Proofs-of-work are utilised as a majority-voting mechanism to enforce the ledger's integrity, achieve global consensus, and provide immunity to minority byzantine attacks, by letting the honest majority "out-work" the faulty transactions.

Despite its importance, the difficulty of realising a proof-of-work scheme with better characteristics than the Blockchain is a central problem in cryptocurrency design. Narayanan et al. [28] summarize it in these terms:

"The holy grail would be to design a consensus protocol with is 'naturally' low-variance by rewarding miners a small amount for lower-difficulty puzzles [...] It remains an open question if there is an alternative version of the consensus protocol which would enable mining puzzles without near-instantaneous broadcast of all solutions."

Here we demonstrate an alternative proof-of-work strcuture which is *implicitly collaborative*, rather than competitive, and addresses all of the above challenges with considerable advantages over prior schemes. We give a proof-of-work scheme that: **(1)** Is fully decentralised without "block consolidation". **(2)** Works in an open model, empowering individuals. **(3)** Promotes cooperation over competition. **(4)** Ensures rapid transaction confirmations. **(5)** Gives commensurate rewards for differing efforts. **(6)** Naturally scales with fluctuating activity.

Our framework altogether removes the need for a blockchain, and allows for a lighter system of transaction verifications arranged into a directed graph: The parallelism between different branches of this graph, or partial orders, allows multiple miners to do *useful work* and get rewarded for verifying the same transactions. This simultaneously removes the "global clock" constraint of the blockchain, as it allows miners to work at unequal speeds with unequal resources, while still obtaining rewards for their efforts, and all the while, contributing to the overall system strength.

Transactions as First-class Objects. Without blocks, transactions are promoted to first-class status, with a dual function: *transactional* and *structural.* Accordingly, our transactions consist of:

– **Transactional/monetary** component, representing the user-facing aspect of the transaction—or *payload*— for example; cash, currencies, securities, contracts, etc. Our framework is oblivious to this component, other than for a minimal ability to carry a value for the fee and reward mechanism.

– **Verification/mining** component, representing the systemic aspect of the transaction—its metadata. Our framework relies on this component to build a globally consistent verification graph using proofs-of-work, incentives, and other mechanisms.

Fees and Incentives. Each transaction must post a transaction fee (e.g., proportional to its size): an offering for collection by future transactions that will verify it. Conversely, each transaction must refer to two prior transactions called parents, which must be valid and have fees available for collection. Referencing the parents indicates verification and causes the verifier to collect a certain amount of available fee from them and their ancestors.

Fees are collected from the two parents' oldest ancestors, with fee remaining, first, by walking the directed acyclic graph of those ancestors and selecting any available fee that remains up to a prescribed total amount of fee, based on the amount of work performed, as a function of the transaction's PoW effort compared to its ancestors' total PoW effort.

1. Verification is directed toward recent transactions, since direct validations of older transactions will be forbidden once their fees are exhausted.

2. Higher fee transactions can attract parallel verifiers for rapid affirmation.

3. Non-zero, but low-fee transactions will eventually get included by smaller miners, with the risk of competition decreasing gradually as time moves on, albeit for lower rewards.

4. Any transaction, once collected into the verification graph, will at some point become connected to the ancestor tree of *every* future transaction in the graph, eventually providing maximum validation regardless of fee – a point we refer to as convergence.

5. Invalid transactions, for reasons such as double spending, stale fees, misformatting, and so on, will be weeded out by majority vote of the miners, who are incentivised to refuse to extend the transaction graph from the fault, else risk their transaction not being included.

Our approach greatly simplifies the central difficulty of maintaining global consistency of a partial order by exploiting the economic incentives available within a cryptocurrency. We incentivise miners to work from the top of the graph (to keep it lean) by requiring that parents still have enough available fee left to make eligible parents. Note that the overhead of embedding the verification process within each transaction is minimal in comparison to the transactional data.

General Benefits. Because our graph-based validation and cooperative proof-of-work framework is mostly independent of the "payload" of the transaction itself—other than a way to collect fees, pay fees, and mint coin— we envision that it should be very straightforward to instantiate it with any cryptocurrency functionality of choice. Transforming any existing blockchain-based cryptocurrency into the "same" cryptocurrency with graph-based verification, gives much better scalability and decentralisation by giving everyone an opportunity to profit from their *individual* participation toward securing the network.

1.2 Related Work

Various problems with Bitcoin's model have previously been studied. Karame et al. [19] evaluate the problem of payment verification speed. Miller et al. [25] look at replacing the proof-of-work in Bitcoin with proofs-of-retrievability, in order to mitigate the waste of computational resources, similar to Park et al. [29] who present a proof-of-space alternative and retrieve decentralisation by making mining available, and reducing the wastage of computational effort in the system. Gervais et al. [15] critically analyse the claims of decentralisation in Bitcoin, while Johnson et al. [17] review the incentives for mining pools to engage in underhanded strategies to achieve a competitive advantage. Pass et al. [30] analyse the Bitcoin Blockchain under an asynchronous setting, where users can join and leave the system as required, and scalability improvements are the focal point of work by Sompolinsky and Zohar [33].

In addition to these works, Miller at al. [26] propose puzzles designed specifically so that they cannot be outsourced, thus removing their utility within mining pools. Further, Lewenberg et al. [21] give a scheme that allows for multiple blockchains to emerge at once, in order to capture more transactions using somewhat overlapping proofs-of-work. These latter two proposals are most closely related to our work, in that they attempt to resolve almost identical issues to us by also proposing alternatives to the standard blockchain structure. Whilst interesting ideas, our work takes a different approach in order to incentivise solo work, and removes the "block" construct altogether, which we feel is the natural destination of this line of enquiry.

In the applied space, IOTA [31] also propose a blockchain free, graph based, structure and Ethereum [12] currently implements the GHOST protocol, which allows for fast (10 second) block creation times, and mitigates the problem of orphan blocks — which they call uncles — by allowing them to be referenced by newly created blocks. It incentivizes this behaviour by increasing the reward for the miner of the new block. Our proposal differs considerably from both these approaches, by attempting to offer a secure framework for which it is possible to build block-free decentralised cryptocurrencies.

2 Block-Free Ledger Protocol

We describe the ground rules for a transaction-based fully decentralised ledger. It relies on collaborative proofs-of-work and derived incentives to achieve global consensus, convergence, and timely verification of new transactions.

Definition 1 (Proof-of-Work Scheme, Difficulty and Work) *A Proof-of-Work Scheme is characterised by a function S taking arbitrary strings a, along with some solution string b, where $S(a, b)$ returns either true or false. We say that S has computational difficulty d, and write $S = S_d$ if no p.p.t. entity, given white-box access to S, and allowed to evaluate it on k inputs, outputs a solution b' such that*

$$\mathsf{Pr}[S(a, b') = \mathit{true}] = k \cdot d^{-1} + \mathit{negl}(a),$$

form some negligible function negl. Furthermore, we say that the work of returning a value b such that $S(a, b)$ returns true is equal to d, and write $\mathsf{Work}(S) = d$.

The function parameter d allows us to vary S_d to target a desired difficulty. In practice, the most common proof-of-work schemes used in decentralised cryptocurrencies are based on hash functions, where the difficulty is easily tunable, verification is quick, and inputs can be arbitrary. We purposefully leave this definition loose so as to allow for maximum flexibility in implementation, so long as we can capture how *challenging* it is to form a certain proof-of-work.

Transactions. Transactions perform all roles in our framework: they mint cash, redistribute value, spend money, add fees and—crucially—confirm the legitimacy of previous transactions. To create a transaction, certain information is provided: payment information, a reference to two previous transactions x_l, x_r, the difficulty being solved c, the fee f, the mint m, and a solution value s,

$$x_i = [\mathsf{Payments}_i, x_{l_i}, x_{r_i}, c_i, f_i, m_i, s_i]. \tag{1}$$

The two parents x_l and x_r are mandatory references to two distinct prior transactions whose validity the present transaction is vouching for (and by transitivity, the validity of all of those two transactions' ancestors also). Provided that the new transaction is itself valid, the PoW attached to the new transaction will add onto the cumulative proof of work associated with both of the parent transactions, and likewise strengthen all of their ancestors.

Formally, we can express an ordering relation on the elements, namely the transactions, with respect to a proof-of-work scheme.

Definition 2 (Transaction Ordering) *Let P be a set of elements called* transactions*. For t and t', two distinct elements in the set, we write $t \prec t'$ if and only if t' contains t within the Proof-of-Work Scheme, and vice-versa for $t \succ t'$. We call the set P equipped with its (partial) ordering relation $\prec$, a Transactional Partially Ordered Set, or T-POSET.*

Definition 3 (Transaction Weight) *Let P be a T-POSET and let $x \in P$ be a transaction or element therein. Let $P' = \{y_i : y_i \succ x\}$ be the set of all the descendants of x. The* weight *of x is defined as the sum of the proof-of-work difficulty contributed by every one of the y_i,*

$$\mathsf{Weight}(x) = \sum_{i=0}^{|P'|} \mathsf{Work}(y_i). \tag{2}$$

Although this notion of *weight* is well defined given a T-POSET, it is dynamic in the sense that as the T-POSET grows with new transactions, the weight of any existing transaction can grow unboundedly as it gains new descendants.

2.1 Fees and Rewards

Every transaction x posts a fee, such that $\mathsf{Fee}(x) > 0$, in order to offset the distributed cost of conveying and verifying the transaction.

Fees are not designed to be immediately passed on to the next claiming transaction, but rather fees increase the total *prize* value of the transaction, available for partial collection by a number of descendants.

Collection. We *require* that every transaction x, linking to a pair of earlier transactions z_1 and z_2, collects a positive and well defined amount of fees, from the union of z_1, z_2 and all of z_1 and z_2's ancestors. Specifically, for the new transaction x to be valid, it must meet the following two conditions:

1. The fee that x will collect must be available *in full* from the fees that remain in the union of all of x's ancestors.

2. Neither prize of z_1 nor the prize of z_2 may be zero.

Those constraints are important later on, in ensuring that new unverified transactions are validated in priority, and that all valid transactions quickly converge to sharing a common descendant.

The amount of fee that a new transaction x is required to collect is fully determined by x and its *local context* (i.e., the smallest subset $P' \subseteq P$ containing x and its ancestors). The collected fee increases monotonically with the difficulty of x's proof-of-work contribution, or even proportionally with an automatic proportion factor β; see Equation 6 and 7.

Prize. Intuitively, the prize of a transaction x w.r.t. P, is the total fee that is still available, from x and all of x's ancestors, for all of x's future descendants (not yet in P). Prize is a dynamic notion: it is highest when x is new and has no descendant, and monotonically decreases as the graph P

grows and the fees from x and its ancestors are picked up by x's descendants. (The prize of x is a well-defined quantity given the current state of P.)

$\mathsf{Prize}_P(x)$ is a very important quantity for a verifier to keep track of, because that is the most fee that a new transaction choosing to verify x will be able to collect from x and its ancestors.

In this context, we call *pass-through* the contribution to the prize of a new transaction, that is "passing through" from the prizes of its combined parents. A hypothetical transaction x that neither paid nor collected any fee would thus have a prize equal to that of its parents, consisting entirely of pass-through. Prize tends to increase from ancestors to descendants, an important property for us.

Depletion. As fees deplete, we need to calculate from which ancestors a new transaction will collect its fees, and how the collection will be apportioned.

The method we employ is to deplete the oldest eligible nodes first. This has two desirable purposes: (1) verifiers will be further compelled to work on newer rather than older transactions, lest their effort risk being for naught; (2) the verification algorithm will be more streamlined, since the sooner the prize of a node reaches zero, the sooner it and all of its ancestors can be pruned from the dynamic data structure that each verifier must maintain to keep track of the amounts still available for collection.

To define the depletion ordering unambiguously, we shall say that *x is older than y* if and only if $\mathsf{Weight}(x) < \mathsf{Weight}(y)$. This relation induces a total order that is compatible with the partial order of the T-POSET. Ties will be highly unlikely, and can be decided in any fixed deterministic manner.

Thus, when a new transaction comes in, the fees that it collects will be garnered from the oldest of its ancestors that still have fees available to collect, moving forward as those oldest transactions become depleted. In this context, $\mathsf{Prize}(x)$ of a transaction x is simply the total amount that can still be collected from x and all of its ancestors y_i, and clearly for all such ancestors we have $\mathsf{Prize}(y_i) \leq \mathsf{Prize}(x)$.

Gross and Net. To formalise these notions, we first need the related notion of *gross fee* of a transaction. The gross fee or just "fee" is the amount paid by a transaction, that will be available for future collection by any descendants. We also define the *net cost* or just "cost" of a transaction, incurred by the transactor, as the gross fee minus any ancestor fees collected and new coins minted (see below).

The "drain" of a transaction x, then, is the current portion of the fee of x that has already been collected by the descendants of x. Clearly, $\mathsf{Drain}_P(x) = 0$ for a transaction without descendants, and approaches $\mathsf{Drain}_P(x) =$

$\mathsf{Fee}(x)$ as x acquires descendants that deplete it. Because $\mathsf{Drain}_P(x)$ is a dynamic notion, it depends on the current state of the T-POSET P in the view of a particular verifier at a given time, which we indicate by the subscript P.

Definition 4 (Drain) *Let P be a T-POSET, containing n transactions, form some n. Let $x \in P$ have $m < n$ decedents $\{y_1, y_2, \ldots, y_m\}$ be the descendant transactions of x, and for each y_i, define δ_i to be the fee taken by y_i from x. Then for each transaction, the drain of x is defined by:*

$$\mathsf{Drain}_P(x) = \sum_{i=1}^{m} \delta_i \tag{3}$$

The "prize" of a transaction x is then the total fees brought by x and its ancestors, minus by the total drain from its descendants. Likewise, prize is a dynamic notion that depends on the current T-POSET P, hence the subscript P.

Definition 5 (Prize) *Let P be a T-POSET, containing n ordered transactions, form some n. Let $x \in P$ be a transaction, and let $m < n$ be the ancestors $\{z_1, z_2, \ldots, z_m\}$ in P of x. The prize of a transaction x with respect to P is defined as the sum, over x and all its ancestors, of the fee minus the drain:*

$$\mathsf{Prize}_P(x) = \big(\mathsf{Fee}(x) - \mathsf{Drain}_P(x)\big) + \sum_{i=1}^{m} \big(\mathsf{Fee}(z_i) - \mathsf{Drain}_P(z_i)\big) \tag{4}$$

We define the prize of a set of transactions, $X \subseteq P$ as,

$$\mathsf{Prize}_P(X) = \sum_{\forall x \in X} \mathsf{Prize}_P(x) \tag{5}$$

Automatic Drain Rate Adjustment. Consider the total prize available across all the current transactions in the system, given by $\mathsf{Prize}_P(P)$. Macroscopically, we can control the time it will take for the combined verification effort to deplete this prize completely (and substitute for it a renewed prize made of the new fees posted with the new transactions). This time is the expected "useful" lifetime of a transaction in the T-POSET, before it can no longer be the direct ancestor of a future transaction.

We control this lifetime by adjusting the rate at which a transaction x can drain the fees from its ancestors, in proportion to the difficulty of the proof of work posted by x. Specifically, assuming for a moment that the combined "proving power" of the whole system is constant, then the drain time as a fraction of the system's age, can be estimated using the ratio of the total available prize (converted to difficulty units) over the total work or difficulty of all the proofs since the system's inception, i.e.,

$$\frac{\mathsf{Time\text{-}to\text{-}drain} \quad (\text{sec.})}{\mathsf{Age\text{-}of\text{-}system} \quad (\text{sec.})} = \beta \cdot \frac{\mathsf{Prize}_P(P)}{\sum_{y_i \in P} \mathsf{Work}(y_i)} \tag{6}$$

Here, β (in number of computations per unit of prize) is an exchange rate parameter indicating how much fee is to be collected from its ancestors by a transaction that posts a proof-of-work of unit difficulty. Time-to-drain must be chosen as a global system constant, and selected large enough to give even to the slower clients an opportunity to solve useful puzzles in their own time, e.g., 1 day (see below for a longer discussion).

We can now adjust the parameter β almost endogenously by solving for β in the above equation every so often, say by forcing recomputation of β in every transaction x whose number of ancestors $|\{y_i \prec x\}| = 2^n$, a power of 2. At all other times, new transactions x are required to use the *most recent* value of β from any ancestor of the transactions it references—where *most recent* refers to the total order induced by the notion of Weight, as defined earlier.

Unfortunately, β determined from the above equation will be technically ill-defined unless all the verifiers share the exact same view of P at the time it is determined. To avoid this problem, we are going to solve for β in a slightly different equation which uses only well-defined inputs. Letting $P' \subset P$ be the uniquely defined set of x's ancestors, we use:

$$\frac{\mathsf{Time\text{-}to\text{-}drain} \quad (\text{sec.})}{\mathsf{Age\text{-}of\text{-}}x \quad (\text{sec.})} = \beta \cdot \frac{\mathsf{Prize}_{P'}(P')}{\sum_{y_i \in P'} \mathsf{Work}(y_i)} \tag{7}$$

This leads to well-defined recomputations of β that are easy to verify.

There is still one aspect of this determination of β that requires an external input: Age-of-x, which needs a clock. Since absolute precision is not paramount to determine β, we propose to let whichever client whose onus it is to recompute β use its own clock, and to require that the verifiers accept it unless the clock skew is very substantial, such as more than one hour.

2.1.1 Minting

Minting is the process whereby new "coins" are created with every valid transaction, as an extra reward. More critically, minting is the process whereby the money supply is gradually inflated from its initial supply of zero. Minted coins go directly to the user independently of fees.

Coins are minted when creating a transaction. A user selects a challenge and pays to themself a value. This value is determined from available data before closing the transaction, and calculated as either,

$$\mathsf{Mint}(x) = f(\mathsf{Work}(x)/\mathsf{Weight}(x)) \cdot \sum_{y_i \prec x} \mathsf{Mint}(y_i) \tag{8}$$

for some monotone function f, for example $f(x) = \alpha \cdot x$ for some constant system parameter α.

While our intention is to describe the function f in such a way that it remains as flexible as possible, realistically, for an f that is compatible with the design goals of our framework, we must insist of some decentralised feedback adjustment mechanism for f, similar to that of the value β (Equation 6 and 7). We expand further on the selection of reward function f in Appendix, Section C.4.

2.1.2 Verification

The transaction verification process is intentionally similar to blockchain-based cryptocurrencies, but with a few twists.

All users normally participate in the verification process, meaning that all participants must collect the transactions issued by other nodes, and record the ones that pass the verification procedure. Users also keep a record of valid transactions that can be later given to new participants who wish to join.

Users verify transactions as they receive them. Upon receiving notice of a transaction x, the client first checks that the two previous transactions included within x are acceptable transactions. The next step of validation is to check that the transaction has the correct proof-of-work attached. A final part for verification is to check that the transaction x itself is valid, which requires that it be both *intrinsically correct* or well-formed, and *extrinsically admissible* or valid in the current ledger context. The former condition is a (static) determination whether the transaction could be valid in the smallest possible ledger context that contains it and its ancestors; if that fails it is forever marked as ill-formed. The latter condition means that we have to check for double spending (and a few other conditions such as availability of fees).

In order to check double spending and resolve conflicts, we use a greedy approach that will ensure consensus. Nodes simply take as valid the (well-formed) transaction that (in their view) has the largest amount of work attached to it and its ancestors: a notion we call *height.* This means that all the ancestors must be well-formed and then accepted as valid as well. This rule is then repeated on the remaining well-formed transactions to accept the one of greatest height (and its ancestors), and so on.

In a normal situation (e.g., barring a powerful attack), conflicts will be shallow and confined to the upper fringe of the growing graph. Clearly, the shallower the conflict, the smaller and faster the local revision needed to resolve it. When an intrinsically correct transaction x is marked invalid because of an extrinsic conflict with a previously accepted transaction y, the rejected transaction x may become valid again if a majority of the network favours x over y.This very same situation occurs in Bitcoin when two competing but otherwise valid blocks share the same prior block.

In order to check that a transaction is inherently well-formed, the criterion is simply that it be valid in at least one POSET, namely the smallest possible POSET that contains that transaction together with all the transactions it references and their ancestors. What makes intrinsic correctness important is that it is a permanent, static notion.

Conflicts and Resolution. Verifiers will normally develop slightly differing views of the current state of the system as it evolves, which can be formalised as saying that they will hold different real views P_1, P_2, ..., of some hypothetical true T-POSET P. This is due to transactions taking some time to propagate through the network, and will resolve by itself as long as no conflict arises.

A conflict arises when two or more transactions $x_1 \in P_1$, $x_2 \in P_2$, etc., are published, such that there can be no single P that contains them all. This normally would require a deliberate attack, such as double spending; but this can also happen by accident, for example, when x_1 and x_2 both refer to almost-depleted parents, so that when the first transaction comes in, the second can no longer claim its fee and is therefore rejected. Due to propagation delays on the network, different users may end up resolving conflicts differently in their own view of the system.

Maintaining a consensus across multiple verifiers in our framework requires the formal notion of the *height* of a transaction. For a transaction x, $\mathsf{Height}(x)$ is the total proof-of-work difficulty expended by all the ancestors of x.

Definition 6 (Height) *Let P be a T-POSET and let $x \in P$ be a transaction or element therein. Let $z_1, \ldots, z_m$ be elements in P all ancestors of x, then the* height *of x is the sum of the proof-of-work difficulty contributed by every one of x's ancestors, plus x itself, given by,*

$$\mathsf{Height}(x) = \mathsf{Work}(x) + \sum_{i=1}^{m} \mathsf{Work}(z_i). \tag{9}$$

Algorithm for Consensus. The rule for conflict resolution is simply stated: *The tallest well-formed transaction prevails* (breaking ties deterministically). A new verifier that comes online can share the network's consensus view of the current T-POSET P of valid transactions by applying the following algorithm:

1. Collect all transactions ever posted, flagging all the ill-formed transactions as permanently ignorable.[2]
2. As long as there remain well-formed transactions that have neither been deemed valid or invalid:
   (a) Select the maximum-height or "tallest" well-formed transaction not yet classified, and classify it as *valid* as well as all of its ancestors.
   (b) While doing do, mark as *invalid* any other transaction that conflicts with any of the newly validated ones.

In Section 2.2 we present a consensus mechanism algorithm that runs in a bounded number of steps, showing that any two honest verifiers A and B that apply it on the same published transaction data sets S_A and S_B, regardless of collection order, will independently reach the same T-POSET view $P_A = P_B$, where additionally:

– Small conflicts only cause small revisions, in the sense that status swapping between valid and invalid can only occur between conflicting transactions and their ancestors up to the point when they share a common ancestor.

– The conflict-resolution rule is fully *incentive-compatible* with the fee-based mechanism previously described, and indeed provides an additional incentive to build new transactions upward from the current "summit" of the graph: any (correct) transaction that builds up from the current summit will become the new summit and thus necessarily valid, in addition to offering the highest prize amongst its ancestors.

– Verifiers who have been accepting the non-consensus branch of a conflict will soon reconcile with the majority consensus, since the majority will extend the graph from the consensus branch at a faster rate than the dissenters, which guarantees that the summit does (or will) belong to the consensus branch. At that point, all the verifiers will have to accept it.

In practice, an implementation of this consensus strategy will of course have to process additions and conflicts incrementally, which can be done very efficiently for shallow conflicts. Other *implementation considerations* are the possibility of heuristically delaying *"reversal"-causing updates* (i.e., the switching of a well-formed transaction from valid to invalid status, or

[2]Permanently ignorable transactions are those that could *never* become valid, e.g., because they carry an invalid signature, have two or more ancestors that mutually conflict, or carry an illegal transaction payload.

vice versa), in order to amortise their costs, as long as the delay does not stymie the consensus rule.

This conflict resolution mechanism is a POSET-based generalisation of the linear conflict resolution mechanism introduced in the Bitcoin Blockchain.

2.2 Efficient Consensus

Maintaining consensus requires incrementally computing the *height* of every transaction. For an n-node annotated directed acyclic graph, the trivial algorithm to compute the height of a node runs in $O(n)$ time, requiring $O(n^2)$ time to do the same for every node in the graph. Unfortunately, $O(n^2)$ soon becomes far too large for a network with a constantly expanding set of transaction nodes. In order to maintain efficiency we present, in Appendix A, an algorithm which runs in time $O(n^2)$ where n is the number of transactions, on a strict subset of the total number of transactions. Specifically, on a set that has not yet reached a convergence – a property described in Theorem 3.

2.3 Transaction Payload

The framework we present is essentially agnostic as to what comes into the transactions. For example, we could use single-denomination transactions that endorse transfers between public keys, and powered by a scripting language *à la* Bitcoin. Alternatively, one could use Ethereum-like transactions with a richer scripting language.The only difference is that, instead of a hash chain system, transactions would be cross-verifying in a T-POSET structure.

Our only requirement is that transactions provide a peer-to-peer ledger-based *value creation and transfer* mechanism based on digital signatures, that our underlying framework can access in order to implement minting and fees.

3 Security and Properties

We now delve into analysing the system security. First, we need to make some assumptions on the participants, in keeping with the decentralised cryptocurrencies. The proof of all theorems are given in the appendices.

3.1 Rational Players

We assume that a majority of participants act rationally, and that this majority adheres to the correctness rules of the protocol. Incorrect, or invalid, transactions will be weeded out by the honest participants, which ensures integrity among them as long as they are the majority. Equally important is the assumption of *rational* participants (whether they are cheating or not), rather, we assume that the majority of the computing power is held by rational players.

Assumption 1 (Rational Players: Miners and Transactors) *A miner acts rationally if they seek to maximise the value of their reward (from minting and fee collection alike) for any given amount of expended effort. A transactor acts rationally if they seek to ensure the acceptance of transactions in which they are the payer or payee. A rational player takes on both aspects.*

3.2 System Properties

Double-Spending Resistance. A key priority is to ensure that a broadcast transaction quickly becomes agreed on by the majority of nodes, and cannot later be nullified in some way. Depending on its importance, once the transaction has gained enough weight, from the total of its descendants' work, it can be deemed *impassible*: it will no longer be feasible or economical for an adversary to try to displace it.

For any two transactions x and y, we say that x conflicts with y if for any honest party U accounting for incoming transactions, U can accept either x or y but not both. There are a few ways for conflicts to occur, with the most obvious ones being of two transactions attempting to make payments from the same source, and of a transaction attempting to extract too much value from an insufficient source. We refer to both as instances of *double spending.*

Theorem 1 shows that the weight, or total verification work accumulated on a transaction by it and its descendants, directly translates to its level of security.

Theorem 1 (Double-Spending Resistance) *Let P be a T–POSET, let x be a transaction element in P, and x the total weight of x in the context of P. Let A be a p.p.t. challenger attempting to include a transaction y that conflicts with x. Suppose that the total number of computational steps performed by A, is k. Then the probability that A can cause y to displace x in the majority consensus, is non-negligibly no greater than $k \cdot c^{-1}$.*

Leading-Edge Preference. The scheme as envisaged provides an incentive to work on the latest transactions—the leading edge—which is important both for fast verification and for convergence. We show this by appeal to rationality.

Theorem 2 (Leading-Edge Preference) *Let P be a T–POSET, and let $x_1 \in P$ be a legitimate transaction forming a proof-of-work on two other distinct transactions x_2 and x_3, both in P. The optimal strategy for any rational player is to include x_1 within the proof-of-work of its next transaction, over x_2 and x_3.*

Convergence. We now analyse the time it would take for transactions published at a given time to all coincide with a common ancestor. Convergence, in this sense, means that at some point there will be a future transaction, x, which will be a common descendent of all the presently published transactions, provided of course that the transactions of interest are acceptable, valid and non-conflicting. We show that, with rational players, all such transactions will at some point share a common ancestor.

Theorem 3 (Convergence) *Let P be a T–POSET and let there be n published transactions such that all transactions are valid, altogether non-conflicting in P. Assuming rational players with varying computational abilities, after some period t, all n transactions will share a common descendant.*

Strong Convergence. Not only is it the case that any given set of suitable transactions will soon share at least one common descendant, we can also prove that, at some later point, any further transaction will always be a descendant of the entire initial set.

Theorem 4 (Strong Convergence) *Let P be a T–POSET and let $Q \subseteq P$ be a subset of n transactions in P, all valid, non-conflicting, and with no descendants in P. Assuming rational players, for any large enough superset $P' \supset P$, then any future transaction that can be added to P' must be a descendant of all the transactions in Q.*

This shows that the verification graph soon behaves for all practical purposes just as a consolidated blockchain: every new proof of work reaffirming every sufficiently old transaction.

Overview of Attacks. To help unpack the implications of the above theorems, we detail in Appendix C how several common attacks are prevented, as well as detailing the expected stability of the system.

3.3 Implementation

We have created an implementation of our proposal and tested it with transaction rates and delays equivalent to bitcoin. Even without optimisation, our implementation was easily able to handle the throughput. Details can be found in Appendix B.

4 Conclusion

The primary objective of this paper was to establish an alternative way of creating a distributed cryptocurrency that avoids the bottlenecks and centralisation issues that come packaged with blockchain implementations.

We achieved this by redesigning the base layer, in favour of a naturally self-regulating and completely decentralised verification process. We have demonstrated that this process still ensures that all new verification effort secures every previous transaction, after a brief period of convergence for each transaction. We believe this novel design for distributed digital currencies is a valuable improvement for this field of research. By creating a currency in this way, it allows us to get closer to the *moral* of a decentralised system which is still found wanting in current implementations.

References

[1] B-money: `www.weidai.com/bmoney.txt`, [Acc: Feb 17]

[2] Barber, S., Boyen, X., Shi, E., Uzun, E.: Bitter to better - how to make Bitcoin a better currency. In: Financial Cryptography and Data Security – FC (2012)

[3] Ben-Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized anonymous payments from bitcoin. In: IEEE S&P (2014)

[4] Bit-gold: `unenumerated.blogspot.com.au/2005/12/bit-gold.html`, [Acc: Feb 17]

[5] BitcoinXT: `https://bitcoinxt.software/`, [Acc: Feb 17]

[6] Bonneau, J., Miller, A., Clark, J., Narayanan, A., Kroll, J.A., Felten, E.W.: SoK: Research perspectives and challenges for Bitcoin and cryptocurrencies. In: IEEE–S&P (2015)

[7] Chaum, D.: PrivaTegrity: online communication with strong privacy. Presentation at Real-World Crypto 2016, Stanford

[8] Chaum, D., Fiat, A., Naor, M.: Untraceable electronic cash. In: CRYPTO (1988)

[9] Danezis, G., Meiklejohn, S.: Centrally banked cryptocurrencies. In: NDSS (2016)

[10] Dogecoin: `http://dogecoin.com/`, [Acc: May 17]

[11] Dwork, C., Naor, M.: Pricing via processing or combatting junk mail. In: CRYPTO (1993)

[12] Ethereum: `https://www.ethereum.org/`, [Acc: Feb 17]

[13] Eyal, I., Sirer, E.G.: Majority is not enough: Bitcoin mining is vulnerable. In: Financial Cryptography and Data Security - FC (2014)

[14] Garay, J.A., Kiayias, A., Leonardos, N.: The bitcoin backbone protocol: Analysis and applications. In: EUROCRYPT (2015)

[15] Gervais, A., Karame, G.O., Capkun, V., Capkun, S.: Is Bitcoin a decentralized currency? IEEE S&P (2014)

[16] Goodin, D.: Ars Technica. http://arstechnica.com/security/2014/06/bitcoin-security-guarantee-shattered-by-anonymous-miner-with-51-network-power (2014), [Acc: Feb 17]

[17] Johnson, B., Laszka, A., Grossklags, J., Vasek, M., Moore, T.: Game-theoretic analysis of DDoS attacks against bitcoin mining pools. In: Financial Cryptography and Data Security - FC Workshops, BITCOIN (2014)

[18] Juels, A., Brainard, J.G.: Client puzzles: A cryptographic countermeasure against connection depletion attacks. In: NDSS (1999)

[19] Karame, G., Androulaki, E., Capkun, S.: Double-spending fast payments in Bitcoin. In: ACM CCS (2012)

[20] Kosba, A.E., Miller, A., Shi, E., Wen, Z., Papamanthou, C.: Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In: IEEE S&P (2016)

[21] Lewenberg, Y., Sompolinsky, Y., Zohar, A.: Inclusive block chain protocols. In: Financial Cryptography and Data Security - FC (2015)

[22] Litecoin: `https://litecoin.org/`

[23] Meiklejohn, S., Pomarole, M., Jordan, G., Levchenko, K., McCoy, D., Voelker, G.M., Savage, S.: A fistful of bitcoins: characterizing payments among men with no names. Commun. ACM 59(4) (2016)

[24] Miers, I., Garman, C., Green, M., Rubin, A.D.: Zerocoin: Anonymous distributed E-cash from Bitcoin. In: IEEE S&P (2013)

[25] Miller, A., Juels, A., Shi, E., Parno, B., Katz, J.: Permacoin: Repurposing bitcoin work for data preservation. In: IEEE S&P (2014)

[26] Miller, A., Kosba, A.E., Katz, J., Shi, E.: Nonoutsourceable scratch-off puzzles to discourage bitcoin mining coalitions. In: ACM CCS (2015)

[27] Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. `https://bitcoin.org/bitcoin.pdf` (2008)

[28] Narayanan, A., Bonneau, J., Felten, E., Miller, A., Goldfeder, S., Clark, J.: Bitcoin and cryptocurrency technologies, draft (2016)

[29] Park, S., Pietrzak, K., Kwon, A., Alwen, J., Fuchsbauer, G., Gazi, P.: Spacemint: A cryptocurrency based on proofs of space. IACR ePrint (2015)

[30] Pass, R., Seeman, L., Shelat, A.: Analysis of the blockchain protocol in asynchronous networks. IACR ePrint (2016)

[31] Popov, S.: IOTA: The tangle. `http://iotatoken.com/IOTA_Whitepaper.pdf` (2016)

[32] Ripple: `https://ripple.com/`, [Acc: Feb 17]

[33] Sompolinsky, Y., Zohar, A.: Accelerating Bitcoin's transaction processing. fast money grows on trees, not chains. IACR ePrint (2013)

[34] Stebila, D., Kuppusamy, L., Rangasamy, J., Boyd, C., Nieto, J.M.G.: Stronger difficulty notions for client puzzles and denial-of-service-resistant protocols. In: RSA (2011)

[35] Stellar: `https://www.stellar.org/`, [Acc: Feb 17]

[36] Tschorsch, F., Scheuermann, B.: Bitcoin and beyond: A technical survey on decentralized digital currencies. IEEE Communications Surveys and Tutorials (2016)

APPENDIX

A Proposed Algorithm

The basic internal data structure of a transaction in our implementation has the form,

$$x_i = [\mathsf{Payments}, x_l, x_r, f_i, f_o, c][h, d, p][a, d]. \tag{10}$$

This includes the published transaction data: i.e., payment information, references to two previous transactions x_l, x_r, in-fee f_i, out-fee f_o and difficulty c.

The verifier also keeps extra variables for height h, depletion d and prize p, which are not published and indeed evolve dynamically as new transactions come it (except for the height h which depends only on the transaction and its ancestors). Finally, the verifier keeps track of two sets with transactions as elements: PrizeSet and DepletedSet, where PrizeSet is the set of elements with prize remaining, and DepletedSet is the set of elements which are depleted but not converged.

The *add* algorithm, below, is used to process new transactions. It runs in time $O(n + m)$, where n is the number of transactions with prize remaining and m is the number of depleted transactions not yet converged. It is therefore possible to add l new transactions in time $O(l * (n + m + l))$ in the absolute worst case. (Each line of computation is annotated with its complexity.)

```
  /*
  Each transaction in the local user's current POSET view belongs to one
  of:
  PrizeSet ← {...}                              ▷ Implemented as a binary tree
  DepletedSet ← {...}                           ▷ Implemented as a binary tree
  ConvergedDepletedSet ← {...}                  ▷ Implemented as a vector
  */

$Add(x_i : \{x_l, x_r, f_i, f_o, c\})$      ▷ Let $n \leftarrow |PrizeSet|$ and $m \leftarrow |DepletedSet|$
  $x_i.a \leftarrow x_l.a \cup x_r.a \cup \{x_i\}$                    ▷ $O(n + m)$
  for $x_j \in x_i.a \cap PrizeSet$ do
      $x_j.d \leftarrow x_j.d \cup \{x_i\}$                        ▷ $O(n + m)$
  end for
  $x_i.h \leftarrow x_l.h + x_i.c + \sum_{x_j \in x_r.a \setminus x_l.a} x_j.c$      ▷ $O(n + m)$
  $x_i.p \leftarrow \sum_{x_j \in x_i.a} x_j.f_o - x_j.d$               ▷ $O(n + m)$
  if $\{x_l, x_r\} \in$ PrizeSet and $x_i.p \geq x_i.f_i$ and otherwise valid then      ▷
  $O(\log n)$
      PrizeSet ← PrizeSet $\cup \{X_i\}$                    ▷ $O(\log n)$
      $temp \leftarrow x_i.f_i$
      $j \leftarrow 0$
      while $temp > 0$ do
          $feeClaimed \leftarrow min(temp, x_i.a[j].f_o - x_i.a[j].d)$
```

```
            $x_i.a[j].d \leftarrow x_i.a[j].d + feeClaimed$
            $temp \leftarrow temp - feeClaimed$
            **for** $x_d \in x_i.a[j].d$ **do**
                $x_d.p \leftarrow x_d.p - min(x_d.p, feeClaimed)$                    $\triangleright\ O(n+m)$
            **end for**
            **if** $x_i.a[j].p = 0$ **then**
                DepletedSet $\leftarrow$ DepletedSet $\cup \{x_i.a[j]\}$                $\triangleright\ O(\log m)$
                PrizeSet $\leftarrow$ PrizeSet$\backslash\{x_i.a[j]\}$                   $\triangleright\ O(\log n)$
                $x_i.a[j].a \leftarrow \{\}$
                $x_i.a[j].d \leftarrow \{\}$
            **end if**
            j $\leftarrow$ j+1
        **end while**
    **else**
        If necessary reset local view based on greatest height.
    **end if**
```

The *clean* algorithm, below, is periodically used to clean up the data structures of all transactions that both have converged and are depleted, and which therefore no longer need to be tracked individually. It runs in time $O(n(n+m))$.

```
$Clean()$
    $converged \leftarrow \bigcap_{x_j \in PrizeSet} x_j.a$                              $\triangleright\ O(n(n+m))$
    $converged \leftarrow converged \cap DepletedSet$                   $\triangleright\ O(n+m)$
    **for** $x_j \in converged$ **do**
        DepletedSet $\leftarrow$ DepletedSet$\backslash\{x_j\}$                       $\triangleright\ O(log(m))$
        ConvergedDepletedSet $\leftarrow$ ConvergedDepletedSet $\cup \{x_j\}$        $\triangleright\ O(1)$
    **end for**
    **for** $x_j \in PrizeSet$ **do**
        $x_j.a \leftarrow x_j.a\backslash converged$                              $\triangleright\ O(n(n+m))$
    **end for**
```

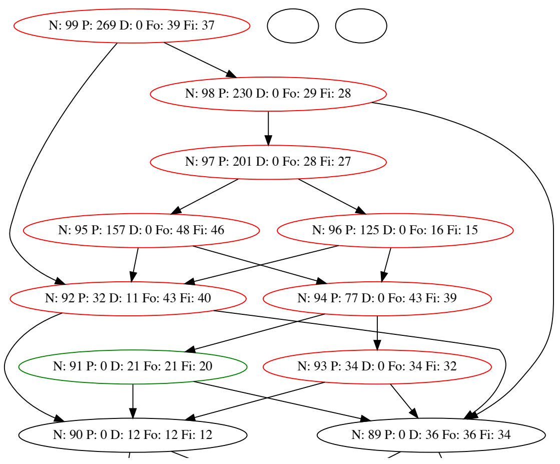
N: 99 P: 269 D: 0 Fo: 39 Fi: 37
N: 98 P: 230 D: 0 Fo: 29 Fi: 28
N: 97 P: 201 D: 0 Fo: 28 Fi: 27
N: 95 P: 157 D: 0 Fo: 48 Fi: 46
N: 96 P: 125 D: 0 Fo: 16 Fi: 15
N: 92 P: 32 D: 11 Fo: 43 Fi: 40
N: 94 P: 77 D: 0 Fo: 43 Fi: 39
N: 91 P: 0 D: 21 Fo: 21 Fi: 20
N: 93 P: 34 D: 0 Fo: 34 Fi: 32
N: 90 P: 0 D: 12 Fo: 12 Fi: 12
N: 89 P: 0 D: 36 Fo: 36 Fi: 34

Figure 3: Top of the graph.

B Implementation and Simulation

We have implemented a network simulator in order to ascertain experimentally that the above verification algorithms induce the right system behaviour. Our implementation is written in C++.

One interesting observation is that the whole converges very quickly even if, whenever new simulated transactions are created, their parents are chosen randomly (amongst the valid choices) instead of game-theoretically to maximise the likelihood of collecting a fee. (The reason we first tried this in the simulation in the first place, is to be able to simulate larger network given the available computing resources.)

Figure 3 and Figure 4 show examples of the resulting transaction graph for a (tiny) 100-transaction example. The nodes in red are those still with a remaining non-zero available prize; the nodes in green are the nodes that have depleted; and the nodes in black are those that have depleted and also converged, meaning that they will necessarily be ancestors of any and all new valid transaction added to this graph.

Table 1 collects statistics from the simulations. Of particularly interest are the last two rows (data rows 6 and 7) where the simulation has been tuned to have an equivalent transaction rate as the current Bitcoin network[3], adjusted for simulation length in order to achieve the same median network delay[4]. The upshot is that our simulation, running on a single i7-4770 CPU (with no optimisation) was easily able to match the transaction rate of the current Bitcoin network and verify all those transactions as they came in.

[3]Our reference transaction rate is peak Bitcoin rate of 260,000 transactions/day on Mon 2016/11/07. `http://www.coindesk.com/data/bitcoin-daily-transactions`

[4]Median delay on 2016/03/07 per `http://bitcoinstats.com/network/propagation`

Table 1: Simulation statistics for several simulation runs, indicating for each: the simulated time length of the execution, the final number of nodes, the average transaction arrival rate, the incurred delay, the average time to live until a transaction becomes depleted, the average time until convergence, and the real CPU time taken by the simulation. All times are in the simulation clock, except for the CPU time which is in real seconds.

Length	Nodes	Node Rate(avg)	Net. Delay	Time To Live	Converge	Execution Time
10hr	5000	0.14 per s	10s	691s	7s	11s
10hr	10000	0.28 per s	10s	626s	511s	11s
10hr	20000	0.56 per s	10s	752s	3348s	58s
10hr	30000	0.83 per s	5s	48s	162s	87s
10hr	40000	1.11 per s	5s	126s	726s	260s
1hr	10833	3 per s	6.2s	582s	1608s	13s
2hr	21666	3 per s	6.2s	1063s	3154s	63s

C Discussion and and Frequently Asked Question

We devote this section to a discussion of the properties of our system, highlighting the differences with Bitcoin (and generally all Blockchain-based cryptocurrencies).

Some of our system's properties are "soft" properties, e.g., from our incentive structures, which will apply is the participants behave rationally.

Other properties are "hard" properties, which are strictly enforced by the rules of validity; they will always apply.

C.1 Properties.

Stability. As the system progresses, conflicting subgraphs may have appeared and branched out from the main graph. These subgraphs should eventually be discarded to avoid overloading the memory of the verifiers; however, they must persist for some period of time until it has become clear that the network consensus is indeed that those conflicting transactions should be discarded (rather than the transactions they are conflicting with).

For example, if two conflicting transactions are relayed to different nodes, both nodes may later receive the other conflicting transaction. After some period, additional transactions will be broadcast predominantly confirming either one or the other of the conflicting transactions. Eventually, one transaction will pull ahead of the other in terms of weight, and as the difference increases the verifiers will switch to the winner's side, to reach global consensus. (And more forcefully, at some point the summit of the graph will be a descendent of either one of the conflicting transactions, at which point all verifiers will be forced to accept it if they haven't already done so.)

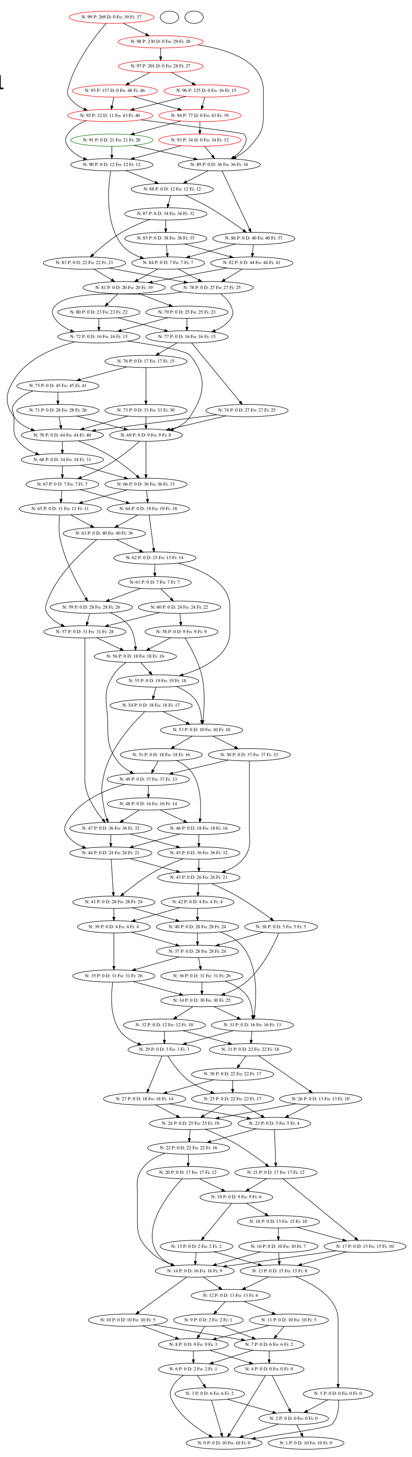
Figure 4: A tiny graph.

Liveness. An immediate consequence of our incentive structure, is that the system will exhibit a *liveness* property, meaning that no, otherwise valid and *compelling*, transaction will stay unverified and thus orphaned for long, before being incorporated into the mesh of converged transactions.

An orphaned transaction is one that has not been verified yet, so by definition its available prize must be strictly positive. As time lapses and no verifiers show up to pick up this transaction, a Laplacian probability argument [5] will induce rational verifiers to conclude that it is increasingly unlikely that a competing verifier will suddenly show up to snatch the prize, thus making verification of the orphaned transaction an increasingly appealing proposition. It suffices that *one* such verifier step up to bring the orphaned transaction into the fold, which is thus a near certainty.

The foregoing argument however excludes orphaned transactions that verifiers may deem *non-compelling*, for example because they have an infinitesimal (yet non-zero) prize, or are too old still to be deemed acceptable.

C.2 Attacks.

One salient difference between the Blockchain and our Transaction Graph verification approach, is the short-term vulnerability to attacks.

Casual attacks are simple and easy, such as someone trying to steal back a payment just made, using double spending.

Bitcoin transactions are *defenceless* against such attacks, until they get picked up onto the Blockchain (taking ≥ 10 minutes in theory, and hours in reality due to congestion, which is only mitigated by paying a large fee). Even the commercial services that, for a fee, offer to *guarantee* not-yet-verified Bitcoin transactions, do no such thing; they merely offer an indemnification *warranty* to the recipient in case a conflicting transactions gets picked up instead.

Our framework closes this opportunity for casual attacks very quickly, because yet-unverified but fee-laden transactions act as a magnet for their immediate parallel verification by multiple users.

Concerted attacks are focused attempts to dislodge a specific transaction, using substantial computing power.

The vulnerability profile of a Bitcoin transaction against concerted attacks is essentially the same as a casual attack: defenceless for a significant period until consolidated, then sharing the strength of the block that picked it up, which then increases as the chain predictably extends from there.

In our system, vulnerability decreases right away as verifications pour in. Partially verified transactions retain temporary exposure to a concerted

[5]Laplace estimated the probability that the Sun would rise tomorrow to be nearly one, based on the fact that it has done so for a great many days already, using a generic Bayesian argument without any physics or other specific domain knowledge.

attack, since a powerful attacker may have the temporary local ability to overpower the honest majority by focusing all of its efforts against one specific target. On the contrary, once a transaction nears or reaches *convergence*, it will be as strongly affirmed as it would be in a Blockchain system of equivalent total verification power.

We note that there is little rational value in using much energy to remove a previous transaction beyond its spend value (e.g., in a double-spending scenario; see Theorem 1), outside of attacks that seek to displace a transaction for ulterior motives.

Disruption and Denial of Service (DoS) are attacks where attackers seek to wreck as much havoc to the system as possible, for example, by flooding the network with multiple small transactions with mutual conflicts, in an attempt to confuse verifiers and clog the network.

Verifiers can employ simple heuristics based on the age and offered prize of a transaction to determine if it holds any value before seeking to determine or revise their validity. This is in fact precisely what out incentive structure recommends. By doing so, the system remains unclogged by flooding, unless the attackers are willing to put in an effort equivalent to a 51% attack.

In Bitcoin, by contrast, the bounded size of the blocks combined with their fixed 10-minute renewal creates a DoS vulnerability not present in our system: it is possible to cause the blocks to fill up by sending many small *valid* transactions, at the only cost of their transaction fee, to clog up legitimate transactions.

Example Attack 1 — the Fork
Attack: Fork the the POSET below some victim transaction by broadcasting a transaction of greater height in a different branch.

Defence: Provided that the first transaction is received by honest clients, who use it as an ancestor of their future transactions, it will be protected not by its own height but the height of its highest descendant. Once the transaction becomes converged (see Theorem 3) all future transactions will strengthen it, and the effort to displace it will have to exceed the total verification power of the entire network from then on.

Example Attack 2 — the Poison
Attack: Create a transaction that points to two conflicting ancestors, or is somehow *intrinsically* bogus in some way.

Defence: Any such transaction will be found to be *permanently ignorable* and marked as such by all verifiers. (The consensus algorithm states: "Permanently ignorable transactions are those that could *never* become valid, e.g., because they carry an invalid signature, *have two or more ancestors that mutually conflict*, or carry an illegal transaction payload"; and the algorithm given to identify such transactions will identify them all, regardless of the nature of the poison.)

C.3 Practical Matters

Transaction Size and Cost. We envisage that any implementation would expect or even require that the cost or fee to be posted by a new transaction, directly reflect the total bit-size of the transaction.

Aside from making the transactor defray the cost of larger transactions, this measure has the added benefit of making it slightly more costly for a transaction to pick parents at considerably different levels (because of the way the parents are encoded relative to each other; see condition 3 under Section 2.2). In turn, this fee differential espouses the fact that it will take longer to verify transactions where the discrepancy between the levels of the immediate ancestors is high.

Choosing whether such a rule should be hard-coded in the system, or softly expected by the verifiers as a natural consequence of the game theoretic incentives, remains an implementation decision.

Bootstrapping and Early Adoption. In order to instantiate the system, it will be necessary to create at a pair of origin transactions, which can come loaded with an initial prize for collection. It will also be necessary to select a minting function, which will determine the rate of inflation and hence the inherent incentives, or disincentives, to early adopters. We do not make any recommendation on the choice of minting function, or parameters set for initilization.

Referring to Two Previous Transactions. We have presented a system where transactions are formed with a proof-of-work which refers to *two* previous transactions. In fact, this may seem a bit rigid, and may not be the ideal solution in certain scenarios.

While we believe that this strikes a balance between burden on the network, such as traffic and verification demand, whilst simultaneously allowing for the building of a proof-of-work system, it can be relaxed if necessary: there is nothing in principle that prevents a design where transactions refer to more than two parent transactions. However, such flexibility seems intuitively unnecessarily costly.

Post-Dated Transactions. By enforcing a notion of verification freshness, our framework disallows post-dating (and ante-dating) by default, unlike Bitcoin. We view this as a useful feature, but note that the permissive behaviour of allowing the specification of "validity dates" can be restored at the transaction payload level, for instance if the payload supports smart contracts.

Scalable Throughput and Responsiveness. Unlike Bitcoin and all other blockchained cryptocurrencies, we place no cap on the number of transactions verifiable in any period of time.

Better yet, since transactions verify each other (in a ratio of 2-to-1), a surge in transactions broadcast will be instantly met with an equal surge in verification response.

We note that Bitcoin is mired in a debate concerning the total size of transactions that can appear in a single block [5], an exclusively Blockchain-model problem.

C.4 Inflationary Measures

Our system is designed to be agnostic to the choice of monetary policy, embodied by the minting function f, the parameter α, and their evolution.

It can be either inflationary or deflationary, and rigidly fixed or adjustable by consensus or using any desired external mechanism. We merely note the following:

- As long as the reward function f is not super-linear, there is no incentive at all for verifiers to join forces and form Bitcoin-style mining pools.
- Sub-linear choices for f would disproportionately reward smaller proofs-of-work, and also further discourage the pooling of effort.
- If the function f or the proportionality parameter α is kept constant throughout the life of the system, then the coin supply will grow as the total verification work, i.e., the time-integral of the total verification power.
- Thus, for constant (unchanging) f and α:
  - if the total work stays constant, e.g., after reaching an adoption plateau, inflation will be linear;
  - if the total work follows Moore's law, then inflation will follow the same exponential growth.
- It is easy to target a different—and almost arbitrary—inflation schedule, by bringing up the system with a decentralised feedback adjustment mechanism for f.

D Theorems

Theorem 1 (Double-Spending Resistance—*Restated*) *Let P be a T–POSET, let x be a transaction element in P, and denote by x the total weight of x in the context of P. Let A be a p.p.t. challenger attempting to include a transaction y that conflicts with x. Suppose that the total number of computational steps performed by A, is k. Then the probability that A can* *cause y to displace x in the majority consensus, is non-negligibly no greater than $k \cdot c^{-1}$.*

From the verification procedure, in order to replace transaction x, A needs to imbue transaction y with a greater total weight than x, as described in Definition 3. From Definition 1 it follows that for k steps, the probability of succeeding is $k \cdot c^{-1} + \text{negl}(k)$ as required, where $\text{negl}(k)$ is a negligible function.

Theorem 2 (Leading-Edge Preference—*Restated*) *Let P be a T–POSET, and let $x_1 \in P$ be a legitimate transaction forming a proof-of-work on two other distinct transactions x_2 and x_3, both in P. The optimal strategy for any rational player is to include x_1 within the proof-of-work of its next transaction, over x_2 and x_3.*

Let v_1, v_2 and v_3 represent the total prize of each transaction x_1, x_2 and x_3. Then by definition $v_1 \geq v_2 + v_3$ in any T-POSET P that is a superset of x_1 and x_1's ancestors (including x_2 and x_3). If x_1 includes a fee then v_1 is strictly greater than $v_2 + v_3$.

In the first case, a rational player will prefer to reference x_1 over x_2 or x_3, as it maximises the prize available for collection, thus offering greater expected reward for the amount of effort.

For the second case, the act of referencing just the one transaction x_1 provides the same total prize as x_2 and x_3 combined, whilst allowing for an extra reference.

Theorem 3 (Convergence—*Restated*) *Let P be a T–POSET and let there be n published transactions such that all transactions are valid, altogether non-conflicting in P. Assuming honest rational players with varying computational abilities, after some period t, all n transactions will share a common descendant.*

For all n transactions, if there is one transaction such that all other transactions are the ancestors of it, then we are done. Otherwise, list all transactions that do not have any descendants. Without loss of generality, assume that there are m distinct transactions of this type. Immediately, no transaction in m can be the descendant of another transaction in m.

Now, compare the potential reward of combining each two distinct transactions. By Theorem 2 we know the optimal selection for a rational participant creating a new transaction is to select two transactions for the m transactions such that the total prize is greatest. Now the list of ancestorless transactions has decreased by $m - 1$. By repeating the same argument, eventually all transactions will be combined, in at most $n - 1$ steps. Which is a worst case bound on the maximum steps to convergence. At this point all transactions from the original set of n transactions will share a common descendant.

Theorem 4 (Strong Convergence—*Restated*) *Let P be a T–POSET and let $Q \subseteq P$ be a subset of n transactions in P, all valid, non-conflicting, and with no descendants in P. Assuming rational players, for any large enough superset $P' \supset P$, then any future transaction that can be added to P' must be a descendant of all the transactions in Q.*

By Theorem 3, all transactions in Q will eventually have a common descendant. At this point, this descendant transaction, along with all other descendant-less transactions will be candidates for inclusion by a following transaction, w.l.o.g, let there be n' of these transactions. Now as in the strategy for proof from Theorem 3, all of these transactions will eventually converge. Hence, after at most $(|Q| - 1) + (n' - 1)$ steps there will be one transaction that is a descendant of all previous, including all of Q. By Theorem 2, any new transaction will be incentivised to include this leading transaction, we call this leading transaction x_Q. Now, after more transactions are added, x_Q will eventually have no claimable prize remaining, thus all rational players will be forced to construct transactions on the descendants of x_Q, containing all of Q as required.

This property shows that, after a while, our T-POSET-based verification graph is as strong as a Bitcoin-style consolidated Blockchain: every new proof-of-work reaffirming every sufficiently old transaction.