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

Xavier Boyen
QUT

Christopher Carr
NTNU & QUT

Thomas Haines
QUT

Abstract

We present a radical solution to the two foremost challenges facing "blockchain"-based cryptocurrencies: (1) "mining pool" oligopolies and (2) incompressibility of delays affecting validation. Both problems stem from the Blockchain mechanism itself, which drives participants into a winner-takes-all global contest that amounts to a low-odds high-variance rewards lottery.

Our proposal strips out the "blocks"-&-"chain" consolidation mechanism, instead repurposing the atomic transactions as the only system objects. A fully distributed proof of work, coupled with progressive and predictable rewards, is efficiently layered on top of the transaction structure. Without blocks, the cryptographic "chain" of transaction affirmations turns into a directed graph, whose sparseness, timely growth and global convergence are steered by game-theoretic incentives.

The transaction affirmation process is *cooperative* (rather than competitive), to entice all participants to work *solitarily* at their own pace, rather than in pools at the pace of a blockchain.

In the absence of blocks, we develop a framework that enjoys better decentralisation, improved responsiveness and natural scalability. Crucially, most of the key features of cryptocurrencies are *transaction-bound* rather than blockchain-bound, and are thus compatible with our framework—e.g., scripting, multi denominations, *smart contracts*, etc.

1 Introduction

Bitcoin is evidently the most successful cryptocurrency to date, with a large user base, considerable media attention and a high market valuation. Earlier cryptographic digital cash schemes tended to revolve around trusted central authorities [5]. Bitcoin reformed this classical view of the cryptocurrency archetype, diverging from the mandatory centralisation by instead presenting a *distributed* cryptocurrency without a pre-ordained authority. Remarkably, it managed to do so using only garden-variety cryptographic primitives (signatures and hash functions), relying instead on innovative distributed systems concepts to achieve most of its desirable properties.

Although it was not the first authority-free cryptocurrency, with proposals like Bit-gold [28] and B-money [27] credited with anteriority [21], Bitcoin is the first cryptocurrency scheme to see widespread adoption. Various factors may account for Bitcoin's success, though there is little question that the decentralised design, and its perceived immunity to government interference, played a significant part in the uptake. As a consequence of the growing awareness of Bitcoin, there is a resurgence of academic interest in cryptographically enforced digital cash, and, more generally, what is now called "Blockchain technology," referring to one of the two key innovations of Bitcoin—the other innovation being transaction scripting [14].

Unfortunately, the lack of an *imposed* authority did not prevent an *emergence* of oligopolies ("mining pools") which would naturally consolidate into an unshakeable monopoly if not for self-restraint ostensibly to avert a crisis of confidence. Indeed, despite intentions, the influx of new users, combined with older users seeking to consolidate their grip on the system, it did not take long for problems to emerge, leading to compromises and arguments that called the fundamentals of the system into question.

Some problems could be easily fixed, while others would have required redesigning central aspects of the system (and sometimes did). Notably, there is a growing body of literature focusing on almost all aspects of Bitcoin, looking at analysing, improving and understanding Bitcoin in light of some of its emerging challenges [1, 3, 9, 15, 16, 17, 26]. On top of this, many practical proposals have arisen, explicitly seeking to overcome various inherent limitations of the system. Proposals range from parameter tweaks, or *altcoins* [31, 33], to complete infrastructure redesigns, albeit often with in-built centralisation [34] and even sovereign escrow systems [4] as "solutions" to the perceived problems. Some of the improvements can run directly on the Bitcoin Blockchain with no modifications, such as *coloured coins.* Improved anonymity [2, 17] and multi-denomination smart contracts [32, 35] are some of the issues that have been addressed with varying success.

Despite considerable effort, some bigger issues still remain, which is concerning for the development of distributed digital cash. Aside from the emergence of *mining pools*, the other critical issue is the incompressible delay required for affirmation of a new transaction. We argue that both problems are inherent to "*blockchain technology*" itself, which is the method used by Bitcoin and most other *decentralised* cryptocurrencies to reach verification consensus.

Decentralised Cryptocurrencies: Principles of Bitcoin. Bitcoin is made of of two parts, a currency protocol layer, and a "backbone" protocol layer, referred to as the Blockchain. Within the Blockchain infrastructure there are effectively two type of objects, the *blocks* and the *transactions.*

Transactions embed all the information necessary for Bitcoin to function. Any participant can create and broadcast transactions to the network. A transaction consists of a number of *inputs* and *outputs*, and indicates how many "bitcoins" from the sum of the inputs are to be transferred to each output. Any output with a positive bitcoin balance then acts as an input in a future transaction.

Transactions, however, carry no authority until they become collected and verified by the system participants, who package them into *blocks* in a *chain-like* fashion. In order to be allowed to create a new block, which comes with two substantial rewards (collection of transaction fees and "free money" minting), users compete to solve a highly difficult computational challenge that refers to the packaged transactions. Once solved, the challenge is easy to verify.

Inputs and outputs are specified by *addresses*, which are derived from anonymous public keys in a signature scheme, created as needed. For a transaction to be valid, all its inputs must be signed with the respective private keys. Naturally, the input and output amounts must match (minus a "transaction fee" that will be collected by the next block creator), and the inputs must not have previously been spent. The network solves the double-spending problem by accepting at most one of the conflicting transactions, but not both. Since a transaction can recombine bitcoins arbitrarily from inputs to outputs, they are fully divisible and fungible. What is more, Bitcoin actually allows inputs and outputs to carry *programmatic scripts* which are executed when verifying a transaction, to determine the action to be performed.

The Bitcoin transactions themselves create a *distributed payment ledger* while the purpose of the Blockchain is to ensure a *global consensus* over this ledger. The payment system provides the incentive mechanisms without which the consensus engine would not be able to operate.

Blockchain Challenges. We have already mentioned the ostensible distributed nature of Bitcoin being frequently called into question. This problem 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 Blockchain, the rewards are few and far between, and the only intrinsically fair method of distributing those rewards is as a high-variance probability distribution—essentially a lottery. Risk-averse participants will coalesce into *mining pools* to reduce that variance, at the cost of abdicating their individual oversight duty, which collectively makes the network more brittle to a variety of attacks, such as the well-known *51% attack* [20] and the "selfish miner" or *33% attack* [8], which in some special cases can become a *25% attack*. Proposals have been made to mitigate this issue, by incentivising participants to defect from pools [19], but they do not eliminate the Blockchain's core problem of allowing only low-odds high-variance fair rewards.

In Bitcoin, a feedback loop on difficulty ensures that *puzzle renewal* occurs every 10 minutes on expectation as a Poisson process. From a miner's point of view, 10-minute renewal is a rapid enough pace that turns any communication delay to the Bitcoin network into a significant disadvantage. Perversely, this also encourages any user interested in mining to delay the propagation of any new block data, except for their own. [1]

From a non-mining point of view, the Blockchain pace is far too slow, as the inflexible schedule of block creation makes transaction verification unbearably long. Some altcoins have experimented with shortening the block creation interval (e.g., Litecoin's 2.5 minutes), but this exacerbates the miners' predicament. Proper verification will always be bounded by the block creation *renewal time*. [2]

Further compounding these delays, the Blockchain encourages contention by hard-capping the block size, hence the transaction rate itself.[3] At the time of writing, the lack of available bandwidth on the Blockchain is allegedly being (ab)used by Bitcoin miners to drive up transaction fees without even having to deliver on speed.

Mining Pools. 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. Indeed, a single one of these monoliths temporarily held absolute majority of the computing power on the network [29]. At the time of writing, those mining pools are mostly based in a single country (China), with just *two* of these pools controlling over 50% of the hashpower of the entire network.[4]

This move away from decentralisation is a widely recognised problem, and is one that has come under scrutiny in the past [15, 19]. On the other hand, some lines of research have sought to give even more power to central authorities in an attempt to solve the scalability and power wastage issues [6]. Unfortunately, none of the current proposals completely deal with a major issue of proof-of-work (PoW) based consensus, in that current PoW systems become prohibitive to new members as the value in the system grows, spurring coalition.

In short, the mining-pool problem is one of economic incentives, and is ineluctably tied

[1]Interestingly, Bitcoin mining pools have in the past voluntarily broken up when overshooting the 50% mark, but they seem to feel no compunction to do so above 33% or 25%.

[2]Ironically, commercial Bitcoin users now mitigate the blockchain delays by entrusting third parties as "payment processors", the very thing that Bitcoin sought to avoid.

[3]Bitcoin's transaction rate is capped because blocks come at a fixed rate and have a fixed maximum size. Block size has incidentally been raised once already, from 4kB to 1MB, and is presently the object of a *fork*, with heated debate pitting individual transactors versus miners and mining pools.

[4]Accurate per `blockchain.info` as at 26 April 2016.

to the “chain” aspect of the Blockchain design. The Blockchain reward structure is simply unsuitable at providing rewards to a large population of miners.

Proofs of Work. The concept of proof of work was developed initially by Dwork and Naor [7] for fighting spam. Applications emerged in other areas, such as client puzzles [12, 25], but the most famous scheme to date remains the Blockchain system itself [20].

A central component of Bitcoin and its derivatives, PoWs are utilised as a majority-voting mechanism to enforce the ledger's integrity, achieve global consensus, and provide immunity to minority byzantine attacks, by allowing the honest majority to “out-work” the faulty transactions. At the same time, proofs of work are tasked with the duty of providing incentives in the form of “fair rewards” to the participants, without adjudication by a central authority.

Despite its importance, there still remains relatively little examination of the underlying design, its properties and fundamental limitations. Indeed, the difficulty of realising a proof of work with characteristics better suited than the Blockchain has been recognised as a central problem in the design of decentralised currencies. Narayanan et al. [21] summarise 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.”*

1.1 Our Approach

Here we demonstrate an alternative backbone proof-of-work scheme which is collaborative rather than competitive, and which we claim successfully addresses all of the above challenges and has considerable advantages over the existing schemes. We show how to construct a PoW scheme that:

1. Is fully decentralised without “block consolidation”.
2. Works on an open model, empowering individuals.
3. Promotes cooperation and demotes competition.
4. Ensures fast and strong transaction confirmations.
5. Gives commensurate rewards for differing efforts.
6. Has natural quiescence, without unneeded broadcasts.
7. Naturally scales up and down with fluctuating activity.

Our proof of work altogether removes the large construct of the Blockchain (which adjudicates transactions into a consolidated total order in the form of blocks). Instead it creates a lighter system of limited partial orders by arranging transaction verifications into a directed graph.

The parallelism between different branches of this graph, or partial orders, allows multiple miners to 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 and with unequal resources, and still get rewarded for their efforts, as they are all 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* (moving funds, settling contracts, and so on) and *structural* (validating prior transactions, processing fees, minting new units of value). Accordingly, a transaction in our framework has two (loosely interacting) components:

- A "transactional/monetary" component, representing the user-facing—or *payload*—aspect of the transaction. Its format depends on the user functionalities that the cryptocurrency seeks to provide (e.g., simple cash, multiple currencies, securities, contracts, etc.). Our framework is mostly oblivious to this component, other than for its minimal ability to carry a value.
- A "verification/mining" component, representing the systemic aspect of the transaction—its *metadata*—, common to all instantiations of our framework. Our framework relies on this component to build a globally consistent verification graph using mechanisms such as proofs of work and associated rewards.

In this framework, each transaction must post a transaction fee (e.g., proportional to size) which is offered for collection by future transactions that will verify it. Indeed and conversely, each transaction must refer to a fixed few (e.g., two) prior transactions, which must be valid and have fees available for collection. Those references indicate *verification* and allow the referring transaction to collect a certain amount of transaction fee from the referred transactions *and their own ancestors.*

Fees and Incentives. Fees are collected by walking the directed acyclic graph of ancestors of the two designated transactions, *starting from the most ancient*, and grabbing any available fee that remains *up to* some total amount of fee calculated by a predetermined formula. This total is a function of the transaction's proof-of-work effort or *hash-energy* compared to the total *hash-energy* of the ancestors, among other factors.

Importantly, we require that only prior transactions from which sufficient fees are still available to collect can be designated as a valid verification target. That is, if based on the hardness of its proof of work and other factors, a new transaction is *owed* a fee of v, and it designates the two prior transactions x_1 and x_2 as verification targets, then it must be the case that the total fee v_1 still available from x_1 and its ancestors, plus the total fee v_2, available from x_2 and its ancestors (excluding duplicates), is equal to or exceeds v.

Emerging Properties. These requirements confer many desirable properties:

1. Verification will be steered toward recent transactions, since direct validations of older transactions will be forbidden once their fees are exhausted;
2. Urgent transactions with a higher fee will 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 (as on the Bitcoin blockchain).
5. Invalid transactions (due to, e.g., double spending, stale fees, misformatting, deliberate attacks, etc.) will be weeded out by majority vote of the honest miners, who will simply refuse to extend the transaction graph downstream from the fault.

Our approach resolves the central difficulty of maintaining global consistency of a partial order by exploiting the economic incentives available in a cryptocurrency. The incentives makes it in the miners' best interest to always try to extend the graph from its "strongest/hardest/latest" point, while also referring to prior transactions that might have not yet been confirmed by others. The requirement that the *full* amount of fee owed to a particular transaction (as determined by a formula) be available from the ancestors it verifies, makes the transaction more concise and easier for miners to verify.

Finally, we note that the overhead of embedding the verification of a couple of prior transactions within each new transaction is minimal, compared to the rest of the data that any useful transaction would require. There will generally be little incentive to post a transaction just to collect fees and mining rewards, rather than combine those with a useful transaction. Setting the number of prior verification to a very small number (two) also has the advantage of forcing miners to create fewer but stronger proofs of work. This increases the total resilience of the system, keeps transactions short, and still leaves small rewards available to small players.

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 user-facing cryptocurrency functionality of choice. Transforming any existing blockchain-based cryptocurrency into the "same" cryptocurrency without blockchain, giving much better scalability and decentralisation toward individual users, by giving everyone an opportunity not only to take part, but to profit from creating proofs of work that they can tailor to suit themselves.

In a nutshell, the Blockchain creates inefficiencies that allows middlemen (mining pool operators) to extract a "tax" (in the form of control over the Blockchain) at the expense of the users and miners alike. By eliminating the Blockchain, the users and miners will benefit from faster transactions and better alignment of fees and rewards to the actual costs.

1.2 Related Work

The main problems we focus on are widely recognised within the world of distributed cryptocurrencies, and there have been various proposals that have sought to address some or both of the problems we have discussed, with varying degrees of success. Karame et al. [13] evaluate the problem of payment verification speed. Miller et al. [18] 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 at al. [22] 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. [10] critically analyse the claims of decentralisation in Bitcoin, while Johnson et al. [11] review the incentives for mining pools to engage in underhanded strategies to achieve a competitive advantage. Pass et al. [23] 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 [24].

In addition to these works, Miller at al. [19] propose puzzles designed specifically so that they cannot be outsourced, thus removing their utility within mining pools. Further, Lewenberg et al. [15] put forward a scheme that allows for multiple blockchains to emerge at once. In order to capture more transactions and allow for proofs of work that somewhat overlap. 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 underlying Blockchain structure. Whilst interesting ideas, our work takes a different approach that incentivise solo work, and removes the "Block" construct altogether, which we feel is the natural destination of this line of enquiry.

2 Block-Free Ledger

This section illustrates a method for constructing a proof-of-work system that is fundamentally collaborative in nature. The idea is simple: instead of grouping individual transactions into "verified" blocks strung into a blockchain, transactions themselves are responsible for vouching for other previous transactions, in a graph-like rather than chain-like structure. Proofs of work are still employed for measuring the *weight* (incremental but also cumulative) of all verifications.

We describe our proposal in two parts. The first part concerns the collaborative proof of work and the way it is used to give rise to an efficient and fully decentralised consensus system. This part can be thought of as a drop-in replacement for the Bitcoin blockchain machinery, with greater decentralisation. Mathematically, we can think of this structure as a partially ordered set (POSET), represented graphically as a directed graph, where each node has a *fan-out* of exactly two edges directed toward two distinct nodes called *sinks*, and an unbounded *fan-in* of arbitrarily many edges incoming from as many *sources*, necessarily all distinct. Each node in this graph represents a singular data structure that forms an individual transaction. Each edge in this graph represents a "verification" relationship, indicating that the source transaction is vouching for the validity of the sink transaction. We emphasise that those edges are about *verification* rather than *payments*, which are the concern of the second part.

The second part specifies the various settlement protocol(s) that can be built on top of the verification structure, while only minimally interacting with it. We describe an instance of such a protocol—for making Bitcoin-like settlement transactions in a single unit of account—along with parameters that causes the system to operate as a digital currency.

2.1 Collaborative Proof Of Work

To help express the intuition behind a *collaborative* proof-of-work system, imagine a challenge-response protocol, where the challenge is to return a solution to a puzzle p, which could be an arbitrary string. The puzzle is to find a solution s, such that $f(p, s)$ returns true, for some fixed function f, where f is chosen so that for any p the easiest way to find an s that makes the function true is by blind search with independent probability $1/c$. We call this c the *difficulty* of the puzzle. Thinking of this in a Bitcoin context, p represents the bulk of a new block to be created on the Blockchain. The variable s represents the *iv* that is selected by the challenger. The function f may be taken as the function that returns true if $H(p, s) = m < 2^{|H|}/c$, i.e., if the puzzle and the initialisation vector hashed through some fixed hash function H yields an output that falls in the $1/c$-th lowest portion of its range. The solution s bears witness that the solver has on average sifted through 2^{c-1} candidates before finding s.

Now, instead of accepting *one* string s as solution of a given challenge p of difficulty c, we may instead accept any number of distinct answers $\{s_i\}$ such that they together bear witness that the solver has searched through the same expected 2^{c-1} candidates in total. More generally, the answers s_i may, but need not, refer to the exact given challenge p. They could refer to a sequence of *derived* challenges, as long as it is clear that the total "proof of work" exhibited by the set of solutions is evidence of a total amount of work performed in response to the given challenge p. Figures 1 and 2 illustrate two possible ways to derive sub-challenges from a given p, in such a way that the solution set constitutes a suitable proof of work for the original p.

By creating puzzles of this form, finding a solution can be achieved in stages, by multiple distinct users, even allowing for completely overlapping solutions. This straightforward way

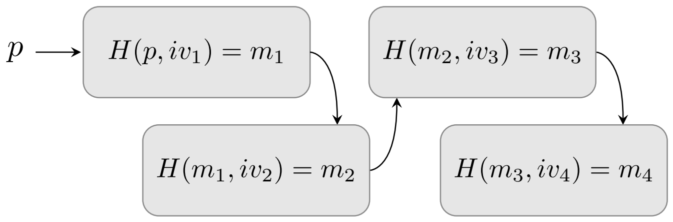

Figure 1: Incremental solution method.

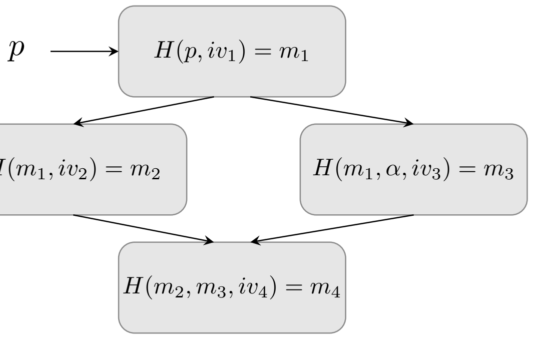

Figure 2: Overlapping solution method.

of accumulating proofs of work establishes the foundations for building our framework, and provides a way of forming a metric on the security of transactions.

What is now needed is a formal method for evaluating the security of the scheme. To achieve this we first need to define define a proof of work scheme and establish a way of capturing challenge form it.

Definition 1 (Proof-of-Work Scheme). *A Proof-of-Work Scheme is characterised by a function $S = S_c$ taking arbitrary strings a, along with some solution string b, where $S(a, b)$ returns either true or false.*

This is a proof-of-work scheme in its simplest form, taking no account for timeliness or succinctness. The function parameter c allows us to vary S_c to target a desired difficulty. In practice, the most common proof-of-work schemes 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. The proof-of-work scheme, to some degree, does not matter so long as we can capture and specify how challenging it is to form a proof.

Definition 2 (Black-Box Difficulty). *A proof-of-work scheme S is said to have difficulty c if any entity, given an arbitrary challenge string a, and one attempt to find a solution b such that the oracle call to $S(a, b)$ returns true, will succeed with probability negligibly close to c^{-1}.*

Definition 3 (White-Box Difficulty). *We further say that S has computational difficulty c if no PPT entity, given open-box access to S, and just enough time to evaluate it on k inputs, outputs a solution b such that $S(a, b)$ with probability non-negligibly greater than kc^{-1}.*

Later on, the term "difficulty" implies the computational notion from Definition 3.

2.2 Protocol

So far we have examined a way of building proofs of work in a collaborative manner. Next, we describe the ground rules of a transaction-based fully decentralised ledger protocol that relies on collaborative proofs of work and derived incentives, to achieve *convergence*, or global consensus and timely verification of new transactions.

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,

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

In practice, the mint m_i can be omitted as it is implied by available data; and likewise the difficulty c_i is implied by the "quality" of the solution seen in the proof-of-work verification.

The two transactions 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 proof of work attached to this new transaction will further strengthen the cumulative proof of work associated with *both* of the two referenced transactions, and in turn, with *all* of their ancestors also.

This gives us a simple but powerful way of "incorporating" new transactions within the system while validating older ones. We can formalise this entire transaction graph as a partially ordered set or POSET, containing within it subsets of strict total orders. For example, returning to Figure 2, one could describe the figure as a POSET with two subset orderings given by: $m_1 \prec m_2 \prec m_4$ and $m_1 \prec m_3 \prec m_4$. For future use, note also that any partial order or POSET can be "lifted" into a compatible total order over the entire set, but this lifting need not be unique. In our example, this would amount to choosing one of $m_2 \prec m_3$ and $m_2 \succ m_3$. For ease of intuition, the POSET we describe can best be visualised as a directed acyclic graph.

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

Definition 4 (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. We call the set P equipped with its (partial) ordering relation $\prec$, a Transactional Partially Ordered Set, or T-POSET.*

We refer to a transaction that is captured in the proof of work of another transaction x as a *parent* transaction. These parent transactions will also contain transactions within their respective proof of work; we call all of these previous transactions recursively captured from the stem transaction x (exclusive of x itself) all the way to the system's inception, as the *ancestors* of x. Respectively, we speak of *children* and *descendant* transactions for the converse notions.

We are now in a position to formally define the *weight* on a transaction, which will become useful in Section 3 for describing validation and security properties.

Definition 5 (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, i.e.,*

$$\mathsf{Weight}(x) = \sum_{y_i \in P'} \mathsf{Work}(y_i). \tag{2}$$

Clearly, this notion of *weight* is a dynamic one, since in a running cryptocurrency the number of descendants of any given transaction can grow unboundedly, with the constant addition of new transactions (and their occasional removal, when there is a tie that must be broken).

This leads us to the monetary and incentive aspects of the proof of work. To operate as desired, our framework relies on two mandatory reward mechanisms—fee collection and coin minting—which must be considered in tandem.

- The fee mechanism is the more complex one, and is designed to steer the rational users to behave for the common good.
- The minting mechanism is simpler and although it also comes with incentives, its principal function is to control the coin supply.

2.2.1 Fees

Every transaction x must post a $\mathsf{Fee}(x) \geq 0$ to offset the distributed cost of conveying and verifying the transaction.

Unlike blockchain-oriented cryptocurrencies, 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. This mechanism is intended to speed up the convergence of the system by providing an incentive for other users to work on newly created transactions rather than any of their ancestors.

The fee paid is arbitrary, but to make the transaction palatable by all verifiers, it is advisable that the fee paid by the new transaction exceed the total fees that it collected from its ancestors. Even so, the transaction may still result in a net profit, from the "minting" component of the reward (discussed later).

Larger fees will entice additional verifiers to contribute to the verification of the newly posted transaction and make it "irreversible" sooner, as previously discussed. Contrast this with the system in Bitcoin, where fees are only a threshold mechanism for including transactions in the Blockchain, without actually speeding up the confirmation process faster than what the pace of the Blockchain can allow.

Collection. In our framework, we *require* every new transaction x that links to a pair of earlier transactions x_1 and x_2, to collect a positive and well determined amount of fees, somehow, from the union of x_1, x_2 and all of x_1 and x_2's ancestors. Specifically, for the new transaction x to be valid, it must meet the following two (or optionally three) 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. In particular, if x_1, x_2 and all their ancestors have been depleted of their fees, then x cannot link to both x_1 and x_2 as the two transactions it wishes to verify.
2. In the updated T-POSET P with the new transaction $x \in P$, the fee now available to collect from x and its ancestors must exceed the fee available for collection from x's ancestors alone. (Using the notion of *prize* defined below, this is to say that $\mathsf{Prize}_P(x) > \mathsf{Prize}_P(\{x_1, x_2]\})$ in the updated T-POSET $P \ni x$.)
3. Optionally, we request (but not require) that the prize of x in the updated T-POSET $P \ni x$, also exceeds the combined prize of x_1 and x_2 in the T-POSET just prior to the update, $P' = P \setminus \{x\}$. (Or in other words, $\mathsf{Prize}_P(x) > \mathsf{Prize}_{P'}(\{x_1, x_2]\})$.)

Those constraints together ensure that new unverified transactions are validated in priority, and that all valid transactions quickly "converge" to sharing a common descendant sooner rather than later.

This mechanism also allows an urgent transaction to be better verified sooner, before reaching "convergence", simply by having it pay a larger fee. This ensures that more fees will be collectable from this transaction in the short term, which in turn will entice additional verifiers to pile on and verify this transaction *in parallel*.

The optional requirement #3 goes beyond #2 by further removing the incentive that a competing verifier might have had to displace the transaction by creating a conflicting transaction.

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^{-1}; see below.

Prize. Intuitively, the prize of a transaction x w.r.t. $P \ni x$, 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 our system.

Depletion. With a new transaction, we need to specify from which ancestor transactions it will collect its fees, and how will the collection assignment 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 broken in any fixed deterministic manner, e.g., by employing a hash function.

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. We now formalise all of 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 its 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 coin 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 reaches $\mathsf{Drain}_P(x) = \mathsf{Fee}(x)$ 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 6 (Dynamic Drain)**.** *Let P be the T-POSET of some universe of valid transactions. The drain of x with respect to P is the sum of all portions of its fee that have been collected by its descendants $z_i \succ x$:*

$$\mathsf{Drain}_P(x) = \sum_{z_i : z_i \succ x} \textsf{\textit{fee from }} x \textsf{\textit{ collected by }} z_i \tag{3}$$

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

Definition 7 (Dynamic Prize)**.** *Let P be the T-POSET of some universe of valid transactions. The prize value 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:*

$$\mathsf{Prize}_P(x) = \sum_{y_i : y_i \preceq x} \Big(\mathsf{Fee}(y_i) - \mathsf{Drain}_P(y_i) \Big) \tag{4}$$

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

$$\mathsf{Prize}_P(X) = \sum_{y_i : y_i \preceq x \textit{ for some } x \in X} \Big(\mathsf{Fee}(y_i) - \mathsf{Drain}_P(y_i) \Big) \tag{5}$$

Note that by summing the gross we avoid double-subtracting the fees collected by y_i itself, which will be counted in $\sum_{z_j \preceq y_i} \mathsf{Drain}_P(z_j)$.

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.,

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

Here, β (in #computations per $$) 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. $\mathsf{Time\text{-}to\text{-}drain}$ is chosen as a global system constant, 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.

Such choice of β is well-defined and locally computable entirely from x and its ancestors. Nevertheless, it ensures that updates to β propagate quickly as earlier transactions *converge* toward common descendants. If a client truly wishes to cheat on the value of β by only referencing older transactions, they will soon be deterred or barred from doing so by our transaction verification obsolescence mechanism.

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:

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

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: $\mathsf{Age\text{-}of\text{-}system}$, which needs a clock. Since precision and accuracy are not paramount for the determination of β, we simply 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, e.g., more than one hour.

2.2.2 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 and "fairly" inflated from its initial supply of zero.

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, e.g., 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, or even,

$$\mathsf{Mint}(x) = \alpha \cdot \frac{\mathsf{Work}(x)}{\mathsf{Weight}(x)} \cdot \sum_{y_i \prec x} \mathsf{Mint}(y_i) \tag{9}$$

for some dimensionless system parameter α.

A lot of flexibility is allowed in the selection of $f(\cdot)$ or α. We briefly mention a few points of interest:

- As long as 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:

  - if the hashing power is constant, inflation is linear;
  - if hashing power 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 or α (similar to that which we described earlier for β).

We emphasize that minted coins go directly to the user independently of fees. Minting may however indirectly affect the balance of incentives to a rational user, depending on the choice of f.

2.2.3 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 the valid transactions so that they can be later given to new participants that wish to join the system.

Users verify transactions as they recieve them. Upon receiving notice of a transaction x, the client first checks that the two previous transactions included within x are acceptable transactions. There are three possibilities:

1. The included transactions have been previously verified and accepted.

2. At least one of the included transactions has not previously been seen.

3. At least one of the included transactions has been seen previously and is invalid.

In other, Bitcoin-like systems, users must also collect and verify all transactions, often without being rewarded for it. Moreover, this step should be relatively quick if the user has already seen and verified multiple 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 simple rule is then repeated on the remaining well-formed transactions to accept the one of highest 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. Either way, the difficulty with conflicts is that, due to propagation delays in the network, different users may end up resolving conflicts differently in their own view of the network.

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 8 (Transaction Height)**.** *Let P be a T-POSET and let $x \in P$ be a transaction or element therein. 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,*

$$\mathsf{Height}(x) = \sum_{y_i \in P : y_i \preceq x} \mathsf{Work}(y_i). \tag{10}$$

Algorithm for Consensus. The rule for conflict resolution is then simply stated as follows: *The tallest well-formed transaction prevails* (breaking ties deterministically). In other words, 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. [5]

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.

[5]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.

We present a consensus mechanism algorithm (Section 2.3) 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 $P_A = P_B$. 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 be enjoined to reconcile with the majority consensus, since the majority will be extending the graph from the consensus branch at a faster rate than the dissenters, which guarantees that the summit does (or will eventually) 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 vice versa), in order to amortise their costs, as long as the delay does not stymie the general consensus rule.

We note that this conflict resolution mechanism is a POSET-based generalisation of the linear conflict resolution mechanism between blocks introduced with the Bitcoin Blockchain.

2.3 Consensus Mechanism

The main difficulty in making consensus highly efficient is that it requires (incrementally) computing the *height* of every transaction. For an n-node annotated directed acyclic graph, the trivial algorithm to compute the sum of any one node's ancestors' value runs in $O(n)$ time, requiring $O(n^2)$ time to do the same for every node in the graph. Unfortunately, no faster algorithm is known for the general problem. Here, we would like to perform the whole-graph computation in $O(n)$ time, with incremental updates in $O(1)$ or at worst $\tilde{O}(1)$ time.

In order to facilitate such a fast update, we place an extra constraint on the topology of the graph by requiring that for any transaction x, its two parents x_l and x_r be "not too distant" from each other. Specifically, we ask that *all the ancestors of x_r after the δ-th generation, also be ancestors of x_l*. For $\delta = 2$, this means that a verifier must check that all $2^\delta = 4$ "grand-parents" of x_r are contained in the ancestry of x_l. For this check to be performed quickly, we require that x provide the (unique) paths from x_l back to each of the 4 grand-parents, in the form of 4 short binary strings compressible into a prefix-tree. Under those conditions, the *height* of x can be determined incrementally as a four-operand addition after verifying the constraint is satisfied. In general, the constraint can be checked in constant $O(|paths|) \leq O(2^\delta)$ time.

To describe this formally, we firstly introduce the notion of *level* of a transaction.

Definition 9 (Transaction Level)**.** *Let P be a T-POSET and let $x \in P$ be a transaction, including two previous transactions x_l and x_r. Then the level of x is given as the maximum of*

x_l and x_r, plus 1,

$$\mathsf{Level}(x) = \max(x_l, x_r) + 1 \tag{11}$$

Unlike the height, the level of a transaction is calculated incrementally in constant time.

Now we state the requirements which must be met by any new transaction posted onto the system. Let P be a T-POSET and let x be a transaction in P, where x links to two previous transactions x_l and x_r.

1. All ancestors of x_r after δ generations must themselves be ancestors of x_l, for some constant design parameter (such as $\delta = 2$).
2. The level of the transaction x_l must be *no less than* the level of the transaction x_r: $\mathsf{Level}(x_l) \geq \mathsf{Level}(x_r)$.
3. Transaction creators must supply a *hint* on how to find, within x_l's binary tree of ancestors, all 2^δ δ-generation ancestors of x_r.

Note that for requirement 1, we are not disallowing x_r from being an ancestor of x_l, however, in that case the height of x will not include any contribution of work from x_r that was not already provided by x_l.

With these additional constraints it is possible to bound the additional effort required to update the heights of the graph and select the correct path. Provided that the three properties are true, then the height of the new transaction x is given as

$$\mathsf{Height}(x) = \mathsf{Height}(x_l) + \sum_{\substack{y \prec x_r \\ y \not\prec x_l}} \mathsf{Work}(y) + \mathsf{Work}(x). \tag{12}$$

Notice that the sum $\sum_y$ contains at most $2^{\delta-1}$ terms, trivially enumerated. In particular, if x_r itself is an ancestor of x_l then the height is simply calculated as $\mathsf{Height}(x) = \mathsf{Height}(x_l) + \mathsf{Work}(x)$.

This algorithm is inherently incremental, and builds from the ground up. This makes the calculation of height at the edge of the system very quick, assuming that the height of x_l is already known.

2.4 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 from/to public keys, and powered by a scripting language *à la* Bitcoin. Alternatively, one could use Stellar or Ethereum-like transactions with a richer scripting language, supporting asset security by the creation of new denominations, and their *trade* using programmatic *smart contracts* enforced by the consensus. 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

Having described the operation of the system, we can now delve into analysing its formal security properties. First, we make some rationality assumptions on the players.

3.1 Rational Players

The first natural but often unstated assumption is that a majority of players follow the correctness rules of the protocol. In our system, as in Bitcoin, 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), and we likewise assume that majority of the computing power is held by rational players. Rationality here comes in two flavours: miner rationality, and transactor rationality, the latter further comprising rationality from the viewpoint of a payer versus that of a payee.

Assumption 1 (Rational Miner). *A miner acts rationally if they seek to maximise the value of their reward (from minting and fee collection alike) for a chosen amount of expended effort.*

Assumption 2 (Rational Transactor). *A transactor acts rationally if they seek to ensure the acceptance of transactions in which they are the payer or payee.*

We now turn to proving the properties of our system.

3.2 Emerging 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 cancelled in some way.

In order to assess this property, we refer to the *weight*, or total proof-of-work difficulty, of a transaction. Depending on its importance, once the transaction has gained enough weight, e.g., from its descendants' work, it can be deemed *impassible*: it will no longer be feasible and/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 (Double-Spending Resistance). *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 PPT 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 $k \cdot c^{-1}$.*

Proof. 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 5. From Definition 3 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 1 is essentially showing that the weight, or total verification work accumulated on a transaction by itself and its descendants, directly translates to the level of security of that transaction.

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 rational mechanisms.

Theorem 2 (Leading-Edge Preference). *Let x_1 be a legitimate transaction forming a proof of work on two other distinct transactions x_2 and x_3. 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.*

The proof relies on the second property of fee collection—namely that the prize of a transaction is always at least as large as that of its parents.

Proof. Let v_1, v_2 and v_3 represent the total value 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 and/or x_3, as it maximises the probability that the required fee will be available for collection when the new transaction is ready, thereby offering greater expected reward for the chosen level 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. $\square$

A stronger property can be made for those transactions that further satisfy property #3—namely that the prize of the new transaction be larger still than the prize of its parents before the new transaction came into existence. As long as this property is true, not only will honest verifiers have an incentive to prefer the new transaction over its parents, but even *dishonest* clients—who might think of actively denying certain valid transactions—will still find it advantageous to prefer the new transaction.

Convergence. We now analyse the time it would take for transactions published at a given time to *converge* onto one another. 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, within this framework with honest 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 honest rational players with varying computational abilities, after some period t, all n transactions will share a common descendent.*

Proof outline. For all n transactions, if all but one transactions are the ancestors of the one transaction then we are done. Otherwise, list all potential transaction parent pairs. We call potential transaction parent pairs, satisfying the global predefined δ condition, *combinable transactions.* Now we can search the list for the combinable transactions with the greatest prize. The rational strategy is to combine these transactions with a new transaction. We include this new transaction and ignore the original two, then start again from the beginning with $n - 1$ transactions. We can do this as it will always be more beneficial to build from the new transaction, from Theorem 2, and because the new transaction has the same ancestors of both of its parents, so it will be combinable with all the transactions that its parents were. By repeating the process 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 will share a common descendant. $\square$

An important point here is to realise that there will never be a point with no combinable transactions. If that happens, then the base of the graph must not be connected, which cannot happen.

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 be a set of n transactions in P, all valid, non-conflicting, and with no descendants in P. Assuming honest rational players, for any large enough superset $P' \supset P$, it is the case than any future transaction that can be added to P' must be a descendant of all the transactions in Q.*

The proof is based on the previous result, which together with our prize-depletion and oldest-fee-first collection mechanisms, ensure that no existing valid transaction in P can continue being verified directly, rather than through its descendants in P', once P' has grown sufficiently large through the action of *rational* verifiers. Using an argument similar to the above, one can then show that any new valid transaction z not in P' but added to P', is necessarily a descendant of at least one $y \in P'$ that is a "shared common descendant" of all the elements of Q. This implies that z is a descendant of all of Q, as required. We omit the formal proof.

This property shows that, "after a while", our T-POSET-based verification graph behaves for all practical purposes just as a Bitcoin-style consolidated Blockchain: every new proof of work reaffirming every sufficiently old transaction.

Stability. As the system progresses, conflicting subgraphs may have appeared and branched out from the main graph to become unused. Of course, 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 to discard those conflicting transactions (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 clearly 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.

4 Discussion

Aside from the direct consequences of our incentive structure on rational users, a number of additional differences to the Blockchain paradigm emerge. For concreteness, we compare mainly to Bitcoin, although the points we make will apply to all Blockchain-based cryptocurrencies.

Attacks. There is a salient difference between the Blockchain and our transaction-based verification approach, in terms of *short-term* vulnerability to attacks.

Casual attacks are simple and easy, such as someone trying to steal back a payment 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).

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. We note that partially verified transactions have temporary exposure to a concerted attack, since a powerful attacker may have the temporary local ability to overpower the honest majority by focusing all of its efforts against a specific target. We note that 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.

There is little value in using energy to remove a previous transaction, outside of attacks that focus on transactions one may wish to remove, such as in a double spend scenario, see Theorem 1.

Disruption and DoS style attacks, where attackers seek to cause as much disturbance to the system as possible, by flooding the network with multiple small transactions, are another potential threat. However, verifiers can employ simple heuristics based on transaction level and the offered prize of a transaction to determine if it holds any value before seeking to determine its validity. Thus the system remains unclogged by flooding, unless the attackers are willing to put in effort equivalent to a 51% attack. This is a contrast to Bitcoin, where small transactions can still be formed to effectively fill blocks, albeit at a cost of transaction fee.

Transaction Size and Cost. We envisage that any system design would make it so that the cost, or fee included with creating a new transaction, would directly reflect the size of the transaction. This measure has the benefit of making it more costly to refer to previous transactions that are on considerably different levels, due to condition three under Section 2.3. This fee reflects the fact it will take longer to verify transactions where the discrepancy between the levels of the immediate ancestors is high. Choosing whether this would be written into the system, or designed so that it naturally evolves as a consequence of the game theoretic incentives, remains an implementation decision.

Bootstrapping. In order to instantiate the system, it will be necessary to create at a pair of origin transactions, which can come loaded with prize for collection, and as many, or as few, as required can be created.

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, and there is nothing in principle preventing a design where transactions refer to more than two parent transactions. However, this protocol would need to be selected depending on what is appropriate for an alternative system.

Early Adoption. In our proposal the choice of minting function will determine the rate of inflation and hence the inherent incentives, or disincentives, to early adopters.

Post-Dated Transactions. By enforcing a notion of verification freshness, our framework disallows this behaviour by default, which we view as a feature, but note that the permissive behaviour can be restored at the transaction payload level.

Scalable Throughput. Unlike 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 met with an equal surge in verification response. We note that Bitcoin is currently mired in a debate concerning the total size of transactions that can appear in a single block [30], an exclusively Blockchain-model problem.

5 Conclusion

We have presented a natural, but radically different, model for distributed digital cash, combining existing technologies and ideas to tackle the biggest issues with contemporary decentralised cryptocurrencies. 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. The graph based proposal maintains the security of the Blockchain while being able to reward every participant for their contribution individually. Moreover, our framework does not appear to have any fundamental drawback, and is universal enough to apply to all of the cryptocurrency application payloads in existence today.

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 found wanting in current implementations.

References

[1] S. Barber, X. Boyen, E. Shi, and E. Uzun. Bitter to better - how to make Bitcoin a better currency. In *Financial Cryptography – FC 2012*.

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

[3] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten. SoK: Research perspectives and challenges for Bitcoin and cryptocurrencies. 2015.

[4] D. Chaum. PrivaTegrity: online communication with strong privacy. Oral presentation at Real-World Crypto 2016, Stanford University, 6 Jan 2016.

[5] D. Chaum, A. Fiat, and M. Naor. Untraceable electronic cash. In *CRYPTO*, 1988.

[6] G. Danezis and S. Meiklejohn. Centrally banked cryptocurrencies. *CoRR*, abs/1505.06895, 2015.

[7] C. Dwork and M. Naor. Pricing via processing or combatting junk mail. 1992.

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

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

[10] A. Gervais, G. O. Karame, V. Capkun, and S. Capkun. Is bitcoin a decentralized currency? *IEEE Security & Privacy*, 12(3), 2014.

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

[12] A. Juels and J. G. Brainard. Client puzzles: A cryptographic countermeasure against connection depletion attacks. In *NDSS*, 1999.

[13] G. Karame, E. Androulaki, and S. Capkun. Double-spending fast payments in bitcoin. In *ACM CCS*, 2012.

[14] A. E. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. *IACR ePrint*, 2015.

[15] Y. Lewenberg, Y. Sompolinsky, and A. Zohar. Inclusive block chain protocols. In *Proceedings of the 22nd ACM*, 2015.

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

[17] I. Miers, C. Garman, M. Green, and A. D. Rubin. Zerocoin: Anonymous distributed e-cash from Bitcoin. In *IEEE Symposium on Security and Privacy*, 2013.

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

[19] A. Miller, A. E. Kosba, J. Katz, and E. Shi. Nonoutsourceable scratch-off puzzles to discourage bitcoin mining coalitions. In *ACM CCS*, 2015.

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

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

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

[23] R. Pass, L. Seeman, and A. Shelat. Analysis of the blockchain protocol in asynchronous networks. *IACR ePrint*, 2016, 2016.

[24] Y. Sompolinsky and A. Zohar. Accelerating bitcoin's transaction processing. fast money grows on trees, not chains. *IACR ePrint*, 2013, 2013.

[25] D. Stebila, L. Kuppusamy, J. Rangasamy, C. Boyd, and J. M. G. Nieto. Stronger difficulty notions for client puzzles and denial-of-service-resistant protocols. In *RSA*, volume 6558 of *LNCS*. Springer, 2011.

[26] F. Tschorsch and B. Scheuermann. Bitcoin and beyond: A technical survey on decentralized digital currencies. *IACR ePrint*, 2015, 2015.

[27] Web. B-money. `www.weidai.com/bmoney.txt`.

[28] Web. Bit-gold. `unenumerated.blogspot.com.au/2005/12/bit-gold.html`.

[29] Web. Dan Goodin Ars Technica. http://arstechnica.com/security/2014/06/bitcoin-security-guarantee-shattered-by-anonymous-miner-with-51-network-power.

[30] Web. BitcoinXT. `https://bitcoinxt.software/`, 2016.

[31] Web. Dogecoin. `http://dogecoin.com/`, 2016.

[32] Web. Ethereum. `https://www.ethereum.org/`, 2016.

[33] Web. Litecoin. `https://litecoin.org/`, 2016.

[34] Web. Ripple. `https://ripple.com/`, 2016.

[35] Web. Stellar. `https://www.stellar.org/`, 2016.