Graphchain: a Blockchain-Free Scalable Decentralised Ledger*

Xavier Boyen†
QUT
Australia
xb@boyen.org

Christopher Carr
NTNU
Norway
chris.carr@item.ntnu.no

Thomas Haines‡
Polyas GmbH
Germany
t.haines@polyas.de

ABSTRACT

Blockchain-based replicated ledgers, pioneered in Bitcoin, are effective against double spending, but inherently attract centralised mining pools and incompressible transaction delays.

We propose a framework that forgoes blockchains, building a decentralised ledger as a self-scaling graph of cross-verifying transactions. New transactions validate prior ones, forming a thin graph secured by a cumulative proof-of-work mechanism giving fair and predictable rewards for each participant.

We exhibit rapid confirmation of new transactions, even across a large network affected by latency. We also show, both theoretically and experimentally, a strong *convergence* property: that any valid transaction entering the system quickly become enshrined in the ancestry of all future transactions.

CCS CONCEPTS

• **Security and privacy** → *Distributed systems security*; • **Theory of computation** → *Cryptographic protocols*; • **Applied computing** → *Digital cash*; • **Software and its engineering** → Massively parallel systems; • **Social and professional topics** → Centralization / decentralization;

KEYWORDS

Cryptocurrencies; Consensus; Decentralisation.

ACM Reference Format:
Xavier Boyen, Christopher Carr, and Thomas Haines. 2018. Graphchain: a Blockchain-Free Scalable Decentralised Ledger. In *BCC'18: The 2nd ACM Workshop on Blockchains, Cryptocurrencies and Contracts, June 4, 2018, Incheon, Republic of Korea.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3205230.3205235

1 INTRODUCTION

Early examples of cryptographic cash relied on central trust [8]. Bitcoin reformed this archetype, offering a "distributed" replicated ledger secured by proof-of-work majority rule without pre-ordained authorities. Alas, the lack of imposed trust did not prevent the emergence of *mining pools*: an oligopoly of syndicates that would consolidate into a monopoly if not for ostensible self-restraint to preserve confidence in the system.

Between the influx of new users, and the hold of older users over the system, it did not take long for questions on the fundamental blockchain design to emerge, pressing for changes. Unfortunately, many issues cannot be fixed without incompatible changes, with proposals ranging from mere tweaks [40, 41], to infrastructure redesigns [1, 4, 16, 23–25, 39], including central control [35] and escrow [7] as putative solutions to the observed problems. Many proposals focus on other issues, such as better anonymity [2, 25] and more expressive smart contracts [30, 43] than the original Bitcoin.

Arguably, two main issues of decentralised cryptocurrencies are: **spurious re-centralisation**, where voting power consolidates into mining pools to make block-based rewards more predictable; and **incompressible verification delays**, inextricably tied to the block renewal time of a blockchain. Both issues are inherent to the blockchain technology itself:

(1) Mining pools and emergent centralisation. This issue is an artefact of the bounded block frequency in a linear blockchain structure, which makes it hard to distribute rewards to the myriad of participants willing to contribute to the verification effort, without what amounts to a high-variance lottery system. Blockchain mining rewards are fair, but too few and far between for solo mining. To ensure a more even income, risk-averse participants coalesce into mining pools. Alas, by doing so, they abdicate their oversight duty, and collectively make the network more brittle to subversion, e.g., via a *51% attack* [28], or a "selfish miner" *33% attack* [15], which with latencies can even be a *25% attack*. Not long ago, a single pool temporarily held an absolute majority of Bitcoin mining power [42]. As syndication negates decentralisation, the lure of pool mining is a widely recognised problem [23, 27].

(2) Verification waiting times. Blockchained cryptocurrencies use a feedback loop to alter mining difficulty to keep the expected block creation time constant, resulting in a stateless Poisson process whose renewal time and standard deviation are on the order of several minutes. For miners, this is barely enough to ensure timely data propagation, and perversely entices them to delay propagating competitors' blocks. For users, the Poisson renewal time acts as a lower bound on the expected transaction verification time—ironically fostering reliance on trusted payment processors for real-time payments, the very thing that Bitcoin sought to avoid.

In short, contention and consolidation are inherent to any blockchain reward structure, making it unsuitable for a large population of miners and users. On the other hand, Bitcoin has demonstrated that proofs of work (PoW) are perhaps the

*A prior version of this work appears at `eprint.iacr.org/2016/871.pdf`
†Xavier Boyen is supported by ARC Future Fellowship FT140101145.
‡Most of this work was done while Thomas Haines was with QUT.
[0]The views and opinions expressed herein are the authors' only.

BCC'18, June 4, 2018, Incheon, Republic of Korea

ACM ISBN 978-1-4503-5758-6/18/06.
https://doi.org/10.1145/3205230.3205235

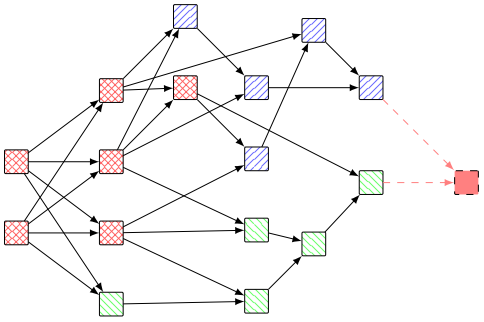

Figure 1: Graphchain of transactions (boxes) with their verification graph (arrows).

most secure way to realise a secure monetary ledger without centralised trust. Accordingly, the question of designing a PoW scheme with better characteristics than a blockchain is a central problem in cryptocurrency design. Paraphrasing Narayanan et al. [29]; the Holy Grail would be to design a (scalable) consensus protocol which is 'naturally' low-variance by offering smaller rewards for lower-difficulty puzzles.

1.1 Our Approach: the Graphchain

We present a blockchain-like decentralised ledger without blocks or a linear chain. The "graphchain" is no longer arranged as a chain of blocks, but as a lean graph formed by tasking each new transaction with confirming some (slightly) older ones. [1] Transactions may post PoWs and get rewarded directly without pooling. All PoWs combine to make the graphchain immutable as a blockchain, but with on-demand scaling and fast response unimpeded by block renewal rate.

A Graph of Cross-Verifying Transactions. Figure 1 shows a tiny set of transactions (boxes) in a directed acyclic *graphchain* of cross-verifications (arrows)—these are *not* the payments flows (not shown). For a transaction to be accepted by peers on a graphchain, it must *verify* a few *parent* transactions, meaning that it vouches for their validity, and their ancestors' by transitivity. Each transaction must have at least two parents, except for a pair of *genesis* transactions which have none. Each transaction also carries a PoW of elective difficulty. This induces a growing graph of confirmations, which cumulatively and transitively affirm all prior valid history.

Efficiency-wise, the graph is steered to grow long and thin, using an incentive system that encourages affirming recent transactions while equitably rewarding all *timely* verifications, even when they happen to share the same parents. Indeed, not being block-based, rewards need not be winner-takes-all, but fairly apportioned, and fostering "competitive collaboration".

Security-wise, after a certain delay, all new transactions will end up indirectly affirming all sufficiently old transactions that are valid, making the graphchain as immutable as in a linear blockchain. We refer to this property as *(strong) convergence*, and say that a set of transactions have (strongly) converged if they all share one (resp. all future) descendant(s) in the graph. In Figure 1, the highlighted transaction is a common descendant of, and thus indirectly verifies, all others. We prove that convergence and strong convergence do occur, and experimentally verify that they occur rapidly, even in a network with delays, which is essential for scalability.

Transactions as First-Class Objects. Standing alone, transactions have a dual aspect serving complementary roles:

- A **transactional**, user-facing application aspect, for moving funds, creating securities, settling contracts, etc. Our framework is essentially oblivious to this component, other than a minimal ability to expose a value mechanism (a "coin") needed by the platform.
- A **structural**, mostly hidden platform aspect, representing the systemic aspect of the transaction, used for validating prior transactions, ensuring irreversibility by consensus, fending off adversaries using PoWs, as well as coin minting and processing transaction fees expressed in value units of a user-facing coin.

Fees and Incentives. Each transaction must post a transaction fee: an offering for collection by future transactions that will verify it. Conversely, each transaction must refer to a selection of prior transactions called parents, which must be valid and have enough fees still available for collection. Referencing the parents vouches for them and causes the verifier to collect a certain amount of available fee from them and their ancestors, function an attached PoW difficulty.

Fees are collected from the oldest ancestors that have any left, by walking the directed acyclic graph of those ancestors and selecting any available fee that remains, to reach a prescribed total amount, function of the transaction's PoW effort compared to the total effort of the ancestors. Consequently:

(1) Verification is steered toward fresh transactions, lest direct validation be forbidden once fees are exhausted.
(2) Higher-fee transactions attract strong miners working and rewarded in parallel, for rapid affirmation.
(3) Low-fee transactions will eventually get picked up by a small worker, as the risk of getting suddenly scooped by another worker decreases as time passes.
(4) Any transaction, once collected into the graphchain, will soon be part of the ancestor tree of *every* future transaction—what we call (strong) convergence—, ensuring maximum irreversibility as in a blockchain.
(5) Invalid transactions, whether inherent (e.g., conflicting ancestors), or contextual (e.g., double spending or stale fees), will be weeded out by majority vote of the miners, who have no incentive to extend the graph from a fault.

Our approach greatly simplifies the challenge of maintaining global consistency of a partial order by exploiting the economic incentives available within a cryptocurrency. Miners are rewarded for working at the top of the graph (to keep it lean) by requiring parents to have enough available fee left to be eligible parents. The overhead of embedding PoW and verification data within each transaction is small in comparison to the transactional data.

[1] The graphchain is a flow of verifications, not payments; it is essentially *independent* of the value transfer graph given transaction chronology.

1.2 Contributions

To our knowledge, our graphchain is the first entirely block-free graph-based cryptographic ledger proposal, which is arguably robust to concerted attacks and suitable as a stand-in for a blockchain. We provide a PoW consensus mechanism with considerable advantages. In short, our proposal:

- Encourages decentralisation over mining pools;
- Provides rapid confirmations without block delays;
- Gives direct rewards commensurate to PoW effort;
- Naturally scales with fluctuating activity.

Our framework substitutes for the common single blockchain, a lighter system of transactions arranged into a directed graph. Multiple miners can get rewarded for verifying the same transactions, shifting emphasis from being the first to create a new block, to picking verification targets to maximise pay-off. Miners working at unequal speeds with unequal resources can still obtain steady rewards for their efforts, without pooling, while still contributing to the overall system strength.

A General Framework. Because the graphchain framework is essentially independent of the transacted token(s) themselves (other than as a basic conduit to offer and claim fees, and create the assets (or coins) needed for the same), we envision that the graphchain could serve as a foundation for any cryptocurrency functionality of choice.

Transforming an existing blockchain-based cryptocurrency into the same cryptocurrency with graph-based verification, would greatly improve scalability and decentralization by giving everyone an opportunity to profit from their individual participation toward securing the network.

Being mostly independent of the value layer, our consensus framework is essentially agnostic to inflation, which can be programmed immutably or updated using consensus itself.

1.3 Related Work

Proofs of work were initially suggested by Dwork and Naor [12] to combat spam. Other applications include client puzzles [19, 38], and famously the Bitcoin blockchain itself [28].

Various problems with Bitcoin's model have previously been studied. Karame et al. [20] evaluate the problem of payment verification speed. Miller et al. [26] look at replacing the PoW in Bitcoin with proofs-of-retrievability, in order to mitigate the waste of computational resources, similar to Park et al. [31] who present a proof-of-space alternative. Gervais et al. [17] critically analyze the claims of decentralization in Bitcoin, while Johnson et al. [18] review the incentives for mining pools to engage in underhanded strategies to achieve a competitive advantage. Pass et al. [32] analyze the 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 [37]. Taking the concern for centralization further, Danezis and Meiklejohn [9] consider the best possible outcomes if a partially-centralized authority is assumed.

Additionally, Miller at al. [27] propose computational puzzles that cannot be outsourced, thus removing the ability for users to work together, such as in mining pools. Lewenberg et al. [23] suggest a scheme where multiple blockchains emerge at once, in order to include more transactions. These two proposals attempt to resolve the two issues we focus on here. Our work, however, takes the different approach of changing the underlying blockchain structure, in order to incentivize solo work, and aims to capture the two desired properties simultaneously. Moreover, in direct contrast to Lewenberg et al. [23], this proposal allows for multiple rewards for securing the same transactions.

DAG-based cryptocurrencies are a natural idea, almost as old as Bitcoin itself, but the devil is in the details. One proposal is that of Lerner [22], which, unfortunately, fails to prevent double spending except in a probabilistic sense, and leaves most questions of scalability and security unanswered.

The Tangle [34, 44] is the design of the IOTA coin. Its transactions are "free", because there is no mining, and no proof-of-X security. It assumes an honest (super-)majority of *users* rather than resources; details are scarce, but Sybil attacks seem prevented only by central control and issuance of coins, all pre-mined. Another concern is scalability. When arranging myriad transactions directly into a graph, not in blocks, one faces semi-numerical algorithms, quadratic in the graph size. The worst case may be *statistically* improbable, yet *adversarially* reachable. The Tangle [34] is statistically well-behaved under chance events, but not necessarily so against deliberate action, leaving it at risk of devastating denial-of-service attacks. (In contrast, it is one of our core contributions to design a block-free graph-based distributed ledger that guarantees fast and correct consensus through a combination of incentives and enforcement.) We were unaware of theirs when our original results came out [5]. As an aside, it later appeared that their (custom-design) hash was broken [13].

Subsequent to our original results [5] came Spectre [36] and the work of Bentov et al. [3]. Spectre, like Bitcoin, uses miners to collect transactions into blocks, but allows those blocks to form a directed graph instead of a chain. They show that this can alleviate a theoretical vulnerability, were the block renewal rate significantly to slow down (e.g,, due to abandon). Spectre however still relies on blocks, and dedicated miners to create them, and so can be viewed as part way between the original blockchain and a true block-free ledger. It does not get rid of either transaction rate bottlenecks or mining pool consolidation. (In particular, we note that despite Spectre being quite posterior to the Graphchain [5], it does not appear influenced by either our construction or its motivation.)

An even more recent proposal of Bentov et al. [3], achieves many similar properties, but uses a much more complicated protocol, involving an additional Byzantine agreement layer.

The Hashgraph [45], recently announced with fanfare, also uses a graph-based structure. They also rely on conventional Byzantine agreement protocols, which, absent centralisation, could fail to Sybil attacks (i.e., fake malicious identities); but argue that it can be changed to a permissionless system. Unfortunately, while they propose switching to proofs of work/stake, they fail to include any analysis of how their incentives provide scalability and security in either setting.

Many other approaches than DAGs have been proposed to circumvent blockchain limitations. Emerging answers to transaction throughput issues, are payment channels such as the Lightning Network [33] and Duplex Micropayment Channels [10]. These channels work by creating a permission-based add-on network for handling transactions; they allow on-blockchain commitments to be leveraged to allow for secure off-chain transactions, allowing significantly increased throughput. These are popular because they work with existing blockchains, and could also complement a graphchain.

Decker et al. [11] published an analysis on the relationship between block size, delay and security in Bitcoin. To rectify some of Bitcoin's delay problems, Sompolinsky et al. [37] introduced the GHOST protocol. While it fixes some issues, it does so by allowing the inclusion of non-accepted blocks, which fails to alleviate the low-odds lottery in mining rewards.

Ethereum [30] currently implements a reduced version of the GHOST protocol, which allows for fast block creation times, and mitigates the problem of orphan blocks, which they call uncles, by allowing them to be referenced in newly created blocks. It incentivises this behaviour by increasing the reward for the miner of the new block. In Ethereum however, transactions within uncle blocks are not valid, and so transaction throughput is not increased. More generally, Ethereum's innovations are directed at the user-facing cryptocurrency layer (e.g., contracts), which are essentially orthogonal to our work, which focuses on the underlying consensus machinery.

Bitcoin NG [14] and Byzcoin [21] are proposals for separating leader selection from transactions. The innovations of Bitcoin NG and its successors are mostly geared at improving mining fairness within the original blockchain paradigm.

The work of Carlsten et al. [6] on the instability of Bitcoin based on fees alone implies a long term problem for Bitcoin and many other currencies. We observe that those dire predictions do not directly apply to us since graphchain fees are not claimed outright but over some specified time span.

2 GRAPHCHAIN FRAMEWORK

We describe the rules for a graph-based ledger. It relies on PoW and derived incentives to achieve global consensus, convergence, and timely verification of new transactions.

Transactions. As the only first-class objects, the transactions themselves perform all roles: they mint and transfer value, and, crucially, confer legitimacy to prior transactions.

Transactions consist of user-specified data and graphchain-specific annotations. User data include amounts, references to prior transaction outputs to be used as inputs, signatures, and scripts or contracts (as in Bitcoin and its derivatives). Graphchain data comprise affirmations of *parent* transactions p_i, a fee f, a mint amount m, and a puzzle solution s,

$$x_i = [\mathsf{Payments}_i, p_i, f_i, m_i, s_i]. \tag{1}$$

The parents p_i are mandatory references to at least one (normally two) prior transactions whose validity the present transaction is vouching for (and ancestors, by transitivity). Provided that the new transaction is itself valid, the PoW attached to the new transaction will add onto the cumulative PoW strength associated with the parent transactions, and likewise strengthen all of their ancestors—just as in a blockchain, each new block strengthening all of its ancestors.

Formally, the verification DAG induces a partial order over the elements with respect to the PoW scheme.

Definition 2.1 (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 the proof of t' encompasses/subsumes that of t, 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.

To avoid ambiguity, $t \prec t'$ means that t is an ancestor of t', and hence buried deeper and "more irreversibly" than t'.

Definition 2.2 (Transaction Weight). Let P be a T–POSET and let $x \in P$ be a transaction or element therein. Let $D = \{y_i : y_i \succ x\} \cup \{x\}$ be the set of all the descendants of x in P, and x itself. The *weight* of x in P is defined as the sum of the proof-of-work difficulties contributed by all of them,

$$\mathsf{Weight}_P(x) = \sum_{y \in D} \mathsf{Work}(y). \tag{2}$$

Although the notion of *weight* is well defined for a T–POSET, it is "dynamic" in the sense that as a T–POSET P_1 grows into another T–POSET P_2 with new transactions, the weight of any preserved transaction from P_1 to P_2 and further, can grow unboundedly as it gains new descendants.

Maintaining a consensus across multiple verifiers who may initially disagree, requires the formal notion of the *height* of a transaction. For a transaction x, $\mathsf{Height}(x)$ is the total PoW difficulty expended by all the ancestors of x.

Definition 2.3 (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,

$$\mathsf{Height}(x) = \mathsf{Work}(x) + \sum_{i=1}^{m} \mathsf{Work}(z_i). \tag{3}$$

Unlike its weight, the height of a transaction x is static and necessarily the same in all T–POSETs P_i in which x appears. If that were not the case, one would have two T–POSETs P_1 and P_2 with identical transactions $x_1 = x_2$ but different ancestries, from which one could extract a hash collision.

2.1 Fees and Rewards

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

Fees, Costs and Incentives. As fees are intended to internalise to the transactor the external costs of a transaction, we envisage that they should at least partially reflect the total bit-size of the relevant transmittal, plus a constant portion. Fees must be strictly greater than 0. There is no mandatory minimum fee. It could be a design choice to specify one, e.g., to combat spam; otherwise, market forces will drive the fees.

Any amount of fee posted above what is expected given the transaction size, acts as an extra incentive to have the transaction verified *sooner*, *faster* and by *stronger* verifiers. Indeed, since verifiers must be paid commensurately and *in full* for their work, a natural stratification of verifiers will occur, tacitly allocating new high-fee transactions to fast miners, and older low-fee ones to slower miners. Accordingly, higher fees will sooner attract faster and heavier verification.

This hierarchy of miners follows from a game-theoretic analysis based on the principle that fast changing environments require snappy action. Here, *the younger a transaction, the shortest should one work on its verification*—lest faster competitors scoop up its fees before one's answer is ready. As a result, fresh transactions will only attract powerful miners who can still deliver a powerful verification in a short time.

Prize. Unlike virtually all mining-based cryptocurrencies in existence, graphchain fees are not designed to be wholly claimed by the first successful miner. Rather, fees increase the total *prize* value of the transaction, available for collection by a number of descendants in proportion to their work.

A transaction (set)'s prize $\mathsf{Prize}_P(x)$ or $\mathsf{Prize}_P(\{x_i\})$ is the sum of all fees from x (or the x_i) and all ancestors, avoiding double-counting, that remain available for collection by future descendant(s) not already in the T–POSET P.

Prize is a dynamic notion: it is highest when x is new and has no child, and decreases as the graph P grows and the fees from x and its ancestors are picked up by x's descendants. Prize is a well-defined quantity given a state P (and can thus go back up if P needs to be rolled back to a previous state).

Reward. We require that a new transaction x, electing to verify parent transactions $z_1, ..., z_n$, collect a specific positive amount from the prize of $\{z_1, ..., z_n\}$, as a verification reward. The reward to be collected is fully determined by the difficulty of x's PoW relative to its *local context*, or smallest subset $P' \subseteq P$ on which the existence of x and the $z_1, ..., z_n$ depends (rewards grow with difficulty; see later, Equations 6 and 7).

Collection Eligibility. For a fresh $x \notin P$ with parents $z_1, ..., z_n \in P$ to be potentially admitted in a new state $P' \ni x$, it must meet two *collection eligibility* conditions:

(1) The exact reward must be collected *in full* from the fees remaining available in all of x's ancestors; esp., x's reward must not overshoot $\mathsf{Prize}_P(\{z_1, ..., z_n\})$.
(2) None of the selected parents' prizes may start empty; i.e., $\forall i, \mathsf{Prize}_P(z_i) > 0$, although $\mathsf{Prize}_{P'}(z_i) \geq 0$.

These constraints act as strong incentives to ensure that: (1) new unverified transactions be validated in priority; (2) all valid transactions quickly converge to common descendants; (3) transactions be concise by proscribing depleted parents.

Depletion Strategy. As fees are collected to form rewards, we need to calculate from which ancestor(s) a new transaction will get its reward, and how the collection will be apportioned.

The method we employ is to deplete the oldest eligible nodes first, i.e., the oldest ancestor with non-depleted prize.

This has two desirable purposes: (1) verifiers will be further compelled to work on newer rather than older transactions, since a descendant will never be depleted before its ancestors; (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 cache that each verifier must maintain to keep track of the prizes.

Deterministic Collection. To make the depletion ordering unambiguous, we shall say that x *is older than* y if and only if $\mathsf{Height}(x) < \mathsf{Height}(y)$, breaking ties deterministically. This relation induces a *total order* that is compatible with the partial order of the T–POSET.

When a transaction arrives, the fees that it collects will be garnered from the oldest of its ancestors that still have uncollected fees, moving forward as the oldest ancestors become depleted. As $\mathsf{Prize}(x)$ is the total amount left to be collected from x and all of its ancestors y_i, clearly $\mathsf{Prize}(y_i) \leq \mathsf{Prize}(x)$.

Drain. The various notions above are most easily expressed in terms of the (current) "drain" of a transaction.

The *drain* of a transaction x is the amount of fee from x itself that has already been collected by its descendants in P.

Clearly, $\mathsf{Drain}_P(x)$ starts at 0 for a new transaction x without descendants, stays at 0 as long as x has non-depleted ancestors, and climbs to reach $\mathsf{Drain}_P(x) = \mathsf{Fee}(x)$ when x itself is depleted. If $\{y_1, ..., y_m\}$ are the parents of x, then $\mathsf{Drain}_P(x) = \mathsf{Prize}_P(x) - \mathsf{Prize}_P(\{y_1, ..., y_m\})$.

Definition 2.4 (Drain). Let P be a T–POSET. Let x be a transaction and let $y_1, y_2, ..., y_m$ be all its descendants in P. If we denote by δ_i the fee claimed by y_i from x proper, then:

$$\mathsf{Drain}_P(x) = \sum_{i:y_i \succ x} \delta_i \tag{4}$$

We can now formally re-define the prize of a transaction x (or set X) as the total fees brought by x (or X) and all ancestors, minus their total drain in the current state P.

Definition 2.5 (Prize). Let $X = \{x_1, ..., x_n\}$ be a set of transactions, and $Z = \{z_1, ..., z_m\} = \{z_i : \exists x_j \in X, z_i \prec x_j\}$ the set of their ancestors. The prize of X with respect to P, is defined as the sum of all fees net of drain in $X \cup Z \subseteq P$:

$$\mathsf{Prize}_P(X) = \sum_{t \in X \cup Z} \big(\mathsf{Fee}(t) - \mathsf{Drain}_P(t)\big) \tag{5}$$

The prize of a single transaction x is $\mathsf{Prize}_P(x) = \mathsf{Prize}_P(\{x\})$.

Insertion Window. 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 the total current prize completely (and substitute for it a renewed prize made of the new fees posted with the new transactions). That length of time, Time-to-drain, is the expected useful lifetime of a new transaction added to P in the optimal way to maximise the transaction's lifetime, before the transaction can no longer serve as the direct parent of a future transaction.

Indeed, whereas Bitcoin transactions can in theory linger indefinitely until either confirmed or double-spent, graphchain

transactions expire if their parents' fees get depleted before themselves being verified. (Conversely, after *convergence*, which should occur quickly, a graphchain transaction becomes as solidly woven into the ledger as it can possibly be.) Accordingly, Time-to-drain should be quite larger than the expected *convergence* time of transactions, lest transactions that have not converged before their parents get depleted risk being forever excluded (more on convergence later on).

Drain Rate Adjustment. We control the Time-to-drain insertion window by adjusting the rate at which a transaction x can drain the fees from its ancestors, in proportion to the difficulty of the PoW posted by x. If we assume 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 current prize, as converted to difficulty units, over the total difficulty of all proofs since the system's inception:

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

Solving for β gives an "exchange rate" indicating the PoW difficulty corresponding to the collection of a unit of reward. Time-to-drain can be fixed as a global system constant, if selected large enough to ensure that network delays do not cause new transactions to incur race conditions.

Unfortunately, β determined from the above equation will be technically ill-defined unless all the verifiers always share the exact same view of P. So, instead, we let $P' \subset P$ be the uniquely defined set of x's ancestors, and solve for β in:

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

where Age-at-x is the age of the system at the moment that x was inserted. This leads to well-defined values of β that are easy to update and check for each verified transaction.

External Clock. Age-of-x 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 blatantly inconsistent (e.g., Age-at-x younger than any of its ancestors, or older than the whole system on the verifier's clock time, with some tolerance for minor clock skews).

Reliance on stated clock time provides an opportunity for cheating: a transaction could claim to be more recent (larger Age-at-x) than it is, resulting in a lower β, and with it a lower amount of work needed for a given reward. Fortunately, this problem is self-limiting: on the one hand, Age-at-x will be almost Age-of-system since only recent transactions will have fees left to claim; on the other hand, Age-at-x cannot exceed Age-of-system, which is publicly known data.

2.2 Verification

The transaction verification process is intentionally similar to blockchain-based cryptocurrencies, except with even more complete decentralisation, and some other small differences.

Decentralised Validation. We expect that users verify transactions immediately as they are received. Upon receiving notice of a new transaction x, the client first checks that all previous transactions included within the PoW of x are acceptable. The next check is to ensure the transaction has the correct PoW attached for the prize being claimed. A final part for verification is to check that the transaction x itself is both *inherently correct* and *contextually admissible* with respect to the current ledger, as the verifier sees it.

Inherent correctness is a (static) determination whether the transaction could be valid in the smallest possible ledger context that contains it, its ancestors, and any input transactions involved in the flow of funds, with *their* inputs and ancestors; if that fails it is forever marked as ill-formed.

Contextual validity involves a (dynamic) check for double spending, and the fulfilment of graphchain conditions such as availability of all fees that the transaction is claiming. We emphasise that those conditions are assessed locally by each verifier, based on the view that this verifier has of the transaction history, and the order of transaction arrival.

Conflicts. Verifiers may develop slightly differing views of the current state of the system as it evolves, which can be formalised as saying that they will hold potentially unequal real views $P_1, P_2, \ldots$, of some hypothetical ground-truth T–POSET P. This is due to transactions taking some time to propagate through the network, and will resolve by itself when benign, or through consensus when conflicts arise.

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 would normally require a deliberate attack, such as double spending, or a race condition when two transactions attempt to claim the last sliver of fee from the same parents, with only enough prize for one reward.

Consensus. The prescriptive rule for conflict resolution is to favour the tallest peak, or *apex*. It is simply stated:

Definition 2.6 (Conflict Resolution Rule). The tallest well-formed transaction prevails (breaking ties deterministically).

Therefore, 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 reference 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 well-formed transaction (the next *peak*) not yet classified, and classify it as *valid* as well as all of its ancestors.
   (b) Mark as *invalid* any other transaction that conflicts with any of the newly validated ones.

[2] Ill-formed or intrinsically incorrect transactions are permanently ignorable as they could *never* become valid in any T-POSET large enough to engender them, e.g., because they carry an invalid signature, have two or more ancestors that mutually conflict, or carry an illegal transaction payload. Such transactions may be safely omitted during the initial coming-online routine or at any later time, since by definition all current and future consensus T-POSETs will work without them.

This conflict resolution mechanism is a generalization of the linear conflict resolution mechanism introduced in the Bitcoin blockchain.

Complexity. Maintaining consensus by the above method requires computing the height of every transaction. For a numerically annotated directed acyclic graph with n nodes, 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, and $\Theta(n^2)$ in the worst-case. Unfortunately, $\Theta(n^2)$ quickly becomes too large for a network with a constantly expanding set of transaction nodes, and there is no known way to do the same with sub-quadratic $o(n^2)$ worst-case complexity, either in batch or incrementally.

To overcome this fundamental limitation, in Section C, we present an algorithm which runs in time $O(\tilde{n}^2)$ where n is the number of transactions and $\tilde{n}$ is the size of a much smaller subset. Specifically, $\tilde{n}$ is the size of the relatively small subset of transactions that have neither expired, nor reached convergence yet, a property described in Theorem 3.3.

2.3 Minting

Minting is the process whereby the money supply is gradually inflated from its initial supply of zero. Minted coins go directly to the user/verifier/transactor independently of fees.

Coins are minted when creating a transaction. A user selects a challenge and pays to themself a mint value (which comes in addition to any reward collected from ancestor fees). The mint amount is determined from available data before closing the transaction, calculated as,

$$\mathsf{Mint}(x) = f(\mathsf{Work}(x)/\mathsf{Height}(x)) \tag{8}$$

for some monotone function f where $x > 0$, 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, to be compatible with the design goals of our framework, we must require a decentralized feedback adjustment mechanism for f, similar to that of the value β (Equation 6 and 7).

Definition 2.7 (Proof-of-Work Scheme, Difficulty and Work). A PoW Scheme is characterized 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 a computation budget equivalent to evaluating S on k inputs, outputs a solution b' such that

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

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 PoW schemes used in modern cryptocurrencies are based on hash functions, where the difficulty is easily adjusted, verification is quick, and inputs can be arbitrary. Our definition purposefully maximises implementation flexibility, so long as it is possible to capture how *challenging* it is to form a PoW.

Inflation and Monetary Policy. Our framework is essentially agnostic to the choice of monetary policy, which is embodied by the minting function f, the parameter α, and the mechanism selected to drive their evolution, whereby it is possible to target an essentially arbitrary inflation schedule. We briefly mention a few points of interest:

- Super-linear and sub-linear choices for f would either encourage users to group together, or otherwise disproportionately reward small PoWs.
- If the function f or the parameter α is kept constant throughout the life of the system, then the coin supply will grow as the total verification work, i.e., as the time-integral of the total verification power:
  - if the computational power is constant, the money supply grows linearly, corresponding to a positive inflation rate decreasing toward zero;
  - if computational power follows Moore's law, then inflation and supply are exponential.

2.4 Transaction Application Layer

So far we have focused almost entirely on the transaction creation/verification/propagation/incentive aspects of our framework, with hardly a word on the transaction functionality itself, i.e., its application layer, or "user-facing payload".

The graphchain framework is essentially agnostic as to what appears in transactions and how they are being used. For example, the application layer could be a single-denomination cryptocurrency used to transfer value between public keys and powered by a simple scripting language, very much like Bitcoin, except on a graphchain rather than a blockchain ledger. Alternatively, the application layer could provide a richer Ethereum-like scripting language, designed for smart contracts, among many other possibilities.

The only generic requirement we place on the application layer, is that it provide a (peer-to-peer, ledger-based) *value creation and transfer* mechanism with signature-based authority, with which the fee/reward and minting functions of the graphchain can interface in order to realise those functions.

3 SECURITY AND PROPERTIES

We can now analyse security by appealing to the incentive structures within the system. First, we need to make some assumptions on the participants, in keeping with the decentralized cryptocurrencies. Our analysis focuses on providing incentives to players to cooperate. To that end we model a notion of rational players, who, unlike honest ones, may stretch or deviate from the protocol incentives if it suits them. Adversaries may deviate even further as they seek to disrupt the protocol entirely. The basis of the analysis assumes that the majority of the participants in the system act rationally.

Rational players. A rational player by default adheres to the hard correctness rules of the protocol, but will deviate from the soft rules and exploit the incentives in order to

gain greater personal gain. Rational players take on two aspects: A rational miner seeks to maximize their net income (from minting and fee collection alike) for a given amount of expended effort. A rational transactor seeks to ensure acceptance of transactions in which they are a payer or payee.

Adversaries. An adversary seeks to disrupt the system, and may act in any way they wish.

Assumption 1 (Rational Majority). *The majority of the computing power in a graphchain system is owned by rational players.*

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. Once the transaction has gained enough weight, from the total of its descendants' work, it can be deemed *immovable*: 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 being that two transactions attempt 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 3.1 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 3.1 (Double-Spending Resistance). *Let P be a T–POSET, let x be a transaction element in P, and denote by c the total weight of x in the context of P. Let A be a p.p.t. adversary attempting to include a transaction y that conflicts with x. Suppose that the total number of computational steps performed by A is k. 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 2.2. From Definition 2.7 it follows that for k steps, the probability of succeeding is $k \cdot c^{-1}$ as required. □

This is simply stating that to displace a transaction, based on a system of rational players, you have to "out-work" the previous transaction—which becomes harder and harder as the victim transaction gets weightier as it gains descendants. Next we show that under the incentive structure presented, transactions consolidate into the verification graph quickly.

Liveness. An immediate consequence of our incentive structure is that it entices verifiers to work on the latest transactions—the leading edge—which is important both for fast verification and convergence. We can prove the following:

Theorem 3.2 (Leading-Edge Preference). *Let P be a T–POSET of all valid and non-conflicting transactions. The optimal strategy for any rational player about to create a new transaction x, is to select the new transaction's parents amongst the set X of childless transactions in P.*

Proof. Consider a transaction $x_0 \in X \subset P$ with ancestor set $\{y_1, y_2, ...\} = Y \subset P$. The rational choice is to prefer x_0, over any $y_i \in Y$, as parent of the new transaction x, since by definition $\mathsf{Prize}_P(x_0) \geq \mathsf{Prize}_P(Y) \geq \mathsf{Prize}_P(y_i)$, as this maximises the probability that x can collect its required fee, lest other transactions $x', x'', ...$ scoop up an unpredictable portion of those fees in the meantime. □

An immediate consequence is that the system will exhibit a *liveness* property, meaning that no otherwise valid and *compelling* transaction is expected to stay unverified for long. This is an important property that we call *convergence*.

Convergence. We seek that there be only a short delay after which all transactions published and verified up to a certain point in time, end up sharing a common descendant. Convergence, in this sense, means that at some point there will be a future transaction, x, which will be a common descendant 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 regardless of their computational abilities, all valid transactions will at some point share a common descendant.

Theorem 3.3 (Convergence). *Consider a sequence of T–POSETs $P_0, P_1, P_2, ...$ resulting from the actions of computationally bounded honest rational players over a sufficiently fast and reliable broadcast channel. Then: (1) The sequence tends to a limit, in the sense that there exists a subsequence of growing T–POSETs $P_{i_0} \subset P_{i_1} \subset P_{i_2} ...$ from the original sequence whose transactions have become immutable; (2) For every T–POSET P_i on the limiting sequence, there will exist another T–POSET P_j also on the limiting sequence, such that all transactions in P_i share a common descendant in P_j.*

Proof. We first prove subclaim (2). Let P_i contain n well-formed and non-conflicting transactions, of which k are still childless in P_i. Under the requirement that each transaction must have at least two distinct parents, by Theorem 3.2 the optimal parents for the first new transaction after P_i will be at least two of the k childless transactions in P_i. The resulting graph P_{i+1} now contains $n + 1$ transactions, of which at most $k - 1$ are still childless (the original k, minus 2, plus 1). By induction, after $k - 1$ steps, P_{i+k-1} will only have one childless transaction (the graph having converged to a string) and that last transaction will be a descendant of the k transactions that were childless in P_i, and hence be a descendant of every transaction from P_i. The argument easily generalises to the case where multiple transactions occasionally arrive without "seeing" each other first, or cause conflicts, in which case convergence will require more steps.

We now prove subclaim (1). As long as transactions do not conflict, the sequence of T–POSETs grows monotonically, so that the actual sequence satisfies $P_0 \subset P_1 \subset P_2 ...$ as required. When a conflict occurs, either due to double-spending or prize

depletion, a transaction $x \in P_i$ may end up being displaced for another $y \in P_j$, so that $P_i \not\subset P_j$ is excluded from the limiting sequence. To show that the limiting sequence exists, and grows, we show that rollbacks are one-off events that do not recur indefinitely. Indeed, displacements are negative-sum events, and thus in aggregate are never beneficial to the majority, which will resist them, and by Theorem 3.1 make it increasingly less likely to succeed even when deliberately created by a computational minority of colluding players. □

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 in time, any further transaction will always be a descendant of the entire initial set.

THEOREM 3.4 (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 a majority of 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 every transaction in Q.*

PROOF. By Theorem 3.3, 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 3.3, all of these transactions will eventually converge. Hence, after $(|Q|-1)+(n'-1)$ steps (or more should conflict resolutions kick in) there will be one transaction that is a descendant of all previous, including all of Q. By Theorem 3.2, any new transaction will have an incentive to include this leading transaction x_Q at first, and then the descendants of this x_Q, again as in the proof of Theorem 3.3. Eventually, x_Q will have no claimable prize remaining, at which point all players will be *forced* to construct transactions on the descendants of x_Q, which themselves are all descendants of all of Q, as required. □

No Submarines. Further to both convergence and strong convergence, it is also the case that if a transaction is ever to reach convergence, it must do so *openly* (and quickly). This is to say that it is not feasible, for an attacker, to nurture a secret side-graph of transactions that branches off the public graph at time t, and then fold it back (and its secret descendants) into the public graph at a much later time $t' \gg t$. We call these: *submarines*; and a motivation to create them could be to mine additional coins in secret, perhaps even with a cheaper spoofed PoW coefficient β.

We show that submarines cannot stay submerged for long. Consider the first secret transaction x, where the submarine branches off the public graph at time t. When at a much later time t', the attacker eventually reveals the secret x and its descendants, all of the parents of x will have been depleted, and hence neither x nor its descendants will be accepted as valid transactions—unless a descendant of x manages to become the Height-iest transaction of the whole graph, in which case the revealed transactions will displace most or all of the public graph after the branch point, in what amounts to a revisionist 51% attack.

We remark that the situation is different with "surface submarines"—timely posted compelling transactions that failed to gain descendants before their parents got depleted. Such a transaction could still be verified, if it is still valid in the majority view, for having already claimed its reward before any of its parents got depleted (i.e., itself contributing to their depletion). This situation is however highly unlikely for a timely posted transaction, as surely a rational verifier would have selected it for verification as in Theorem 3.3. In any case, a transaction that does not converge soon enough, quickly becomes irreversibly invalid, as the consensus rule of Definition 2.6 prescriptively ascribes validity to non-conflicting peak transactions and their ancestors, by decreasing height.

Stability. As the system progresses, conflicting subgraphs may have appeared and branched out from the main graph. in consequence of the No-Submarines property, such branched out subgraphs may only be viable if they are out in the open, hence accidental, as opposed to intentionally hidden. These subgraphs are eventually discarded to avoid overloading the memory of the verifiers; however, they must persist for a short period until it has become clear which ones of the conflicting transactions the network consensus is discarding.

For example, if two conflicting transactions are relayed to different nodes, both nodes may later receive notice of the other conflicting transaction, leaving the system temporarily out of consensus for those two transactions. Soon, though, additional transactions will soon be broadcast, that predominantly confirm 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, all verifiers will conform to the winner's side, to reach global consensus. (More forcefully, by strong convergence, at some point the apex of the graph will be a descendant of either one of the conflicting transactions, at which point all verifiers will be forced to accept it as valid if they had not already.)

Scalable Response, Throughput and Strength. Unlike Bitcoin and blockchain cryptocurrencies, there is no natural or artificial cap on the number of transactions verifiable in any period of time. Since each transaction is required to verify two others, a surge in transactions broadcast will rapidly be met with an equal surge in verification response. A graphchain's throughput is only limited by the communication network's bandwidth and latency, and the nodes' computing power. Though there is no notion of block size or rate, a graphchain soon converges to be as strong as a mined linear blockchain.

4 CONCLUSION

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

by redesigning the base layer, in favor of a naturally self-regulating and completely decentralized 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. Creating a cryptocurrency on a graphchain would allow one to get closer to the moral of a fully decentralised medium of exchange which is still found wanting in current implementations.

A ATTACKS AND DISCUSSION

One salient difference between blockchain and graphchain cryptocurrencies, lies in their short-term vulnerability to attacks, both casual and concerted.

Casual attacks are opportunistic and unsophisticated, such as someone trying to steal back a payment just made, by issuing a competing *double-spending* of the amount paid.

Bitcoin transactions are *defenseless* against such attacks, until they get picked up onto the Blockchain (taking 10 minutes in theory, and up to hours in reality due to congestion). 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 payee in case a conflicting transaction gets picked up instead.

The graphchain shuts opportunities for casual attacks very quickly, because a yet-unverified fee-laden transaction acts as a magnet to rush its parallel validation by multiple verifiers.

Concerted attacks are focused attempts to dislodge a specific transaction for ulterior motives, using substantial computing power far in excess of the face value of the target.

The vulnerability profile of a Bitcoin transaction against concerted attacks is essentially the same as a casual attack: defenseless 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 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. However, once the transaction nears or reaches *convergence*, it will be as strongly affirmed as it would be in a blockchain system of equivalent total verification power.

Disruption and denial of service (DoS) are not directed at transactions themselves as much as they seek to wreck havoc on the whole system. For example, a botnet might create a flood of myriad transactions with subtle 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 our 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 which case they can do more than mere DoS).

In Bitcoin, by contrast, the bounded block size compounds the fixed 10-minute renewal to create a DoS vulnerability not present in our system: it is possible to clog up the blocks by posting many small *valid* transactions, at the only cost of their transaction fees—and rumours suggest that some unscrupulous Bitcoin foes may have engaged in exactly that.

Active consensus denial by splitting the graph is an attempt by an attacker to fork the T-POSET, first by simultaneously broadcasting two conflicting transactions of equal height, then actively working to keep both branches level to prevent global consensus.

In the worst case, the network will be exactly split in two, with one half of verifiers selecting one transaction as valid, and the other as invalid. It will then be a race for one faction to increase the height of "its" branch faster than the other. Once one side moves ahead, the entire network will consider this branch to be the true set. In order to keep the system in a non-ordered state, on the edge of the knife, the adversary has the challenge of attempting to prop up the losing branch, faster than the entire network favours the winning one.

B PRACTICAL CONSIDERATIONS

Bootstrapping and Parametrising. In order to launch a new instance of the system, it will be necessary to create a pair of genesis transactions, loaded with an initial prize for collection. It will also be necessary at least to select a minting function and a depletion time, and/or a policy or strategy for updating the same; those will determine the rate of inflation and the economic incentives for the participants, especially the early adopters. We do not make any specific recommendation as our focus is on the general framework.

Post-Dated Transactions. It is sometimes desirable to sign transactions intended to be released much into the future, something that is always possible in Bitcoin. By enforcing a notion of verification freshness, our framework disallows post-dating (and ante-dating) by default, which to our knowledge is unlike every other cryptocurrency protocol. We view *mandatory freshness* as a security feature of our system—as unverified transactions will expire quickly and unambiguously—, but note that the behaviour of post-dated transactions can be enabled at the application layer, for instance if the instantiated system supports smart contracts.

Light/Thin Clients. Though one main benefit of the graphchain is to enable individuals to run "full nodes" *and be rewarded* for the added security that they provide, this may simply not be feasible or practical on mobile devices.

Such *thin clients* are still able to create transactions autonomously, even if they only collect and retain recent not-yet-converged transactions (for purposes of selecting suitable parents) as well as any user transactions relevant to funding (to serve as inputs in the new transaction). A thin client that does not see the entire graph may not spot all conflicts, and is at risk that its transactions be rejected by consensus.

C SKELETON ALGORITHMS

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

$$x_i = [\{\mathsf{Payments}, f_i\}, p, f_o, c][H, D, P][a, d]. \quad (9)$$

This includes the published transaction data, i.e., payment information, verification references to parent transaction p, in-fee f_i, out-fee f_o, and the implicit difficulty c of the PoW.

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 in the verifier's view (except for the height h which is a well-defined static notion). 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. For each element in the PrizeSet, we store the set of its ancestors, a, and descendants d, who are contained in the PrizeSet and the DepletedSet. For convenience, we denote by $x_i.p.a$ the set $x_i.a \backslash \{x_i\}$.

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 : {p, f_i, f_o, c})               ▷ n = |PrizeSet|, m = |DepletedSet|
  x_i.a ← ⋃_{x_j ∈ x_i.p} x_j.a ∪ {x_i}                          ▷ O(n + m)
  for x_j ∈ x_i.a ∩ PrizeSet do
      x_j.d ← x_j.d ∪ {x_i}                                      ▷ O(n + m)
  end for
  x_i.H ← x_i.p_0.H + x_i.c + Σ_{x_j ∈ x_i.p.a \ x_i.p_0.a} x_j.c   ▷ O(n + m)
  x_i.P ← Σ_{x_j ∈ x_i.a} x_j.f_o − x_j.D                        ▷ O(n + m)
  if {x_l, x_r} ∈ PrizeSet ∧ x_i.P ≥ x_i.f_i ∧ othrws. ok then   ▷ O(log n)
      PrizeSet ← PrizeSet ∪ {X_i}                                ▷ O(log n)
      temp ← x_i.f_i
      j ← 0
      while temp > 0 do
          feeClaimed ← min(temp, x_i.a[j].f_o − x_i.a[j].D)
          x_i.a[j].D ← x_i.a[j].D + feeClaimed
          temp ← temp − feeClaimed
          for x_d ∈ x_i.a[j].d do
              x_d.P ← x_d.P − min(x_d.P, feeClaimed)              ▷ O(n + m)
          end for
          if x_i.a[j].P = 0 then
              DepletedSet ← DepletedSet ∪ {x_i.a[j]}             ▷ O(log m)
              PrizeSet ← PrizeSet\{x_i.a[j]}                     ▷ O(log n)
              x_i.a[j].a ← {}
              x_i.a[j].d ← {}
          end if
          j ← j+1
      end while
  else
      If necessary reset local view based on greatest height.
  end if
```

Table 1: Statistics for several simulation runs, indicating for each: the simulated time length of the execution (LEN), the final number of nodes (NO), the average transaction arrival rate (RTE), the incurred delay (DLY), the average time to live until a transaction becomes depleted (ALT), the average time until convergence (ACT), and the real CPU time taken by the simulation (CPUT). All times are in the simulation clock, except for the CPU time which is in real seconds.

LEN	NO	RTE	DLY	ATL	ACT	CPUT
10hr	5000	0.14 per s	10s	1000s	10s	1s
10hr	10000	0.28 per s	10s	1000s	10s	3s
10hr	20000	0.56 per s	10s	1000s	10s	16s
10hr	30000	0.83 per s	5s	500s	5s	2s
10hr	40000	1.11 per s	5s	500s	5s	3s
1hr	10833	3 per s	6.2s	620s	6.2s	31s
2hr	21666	3 per s	6.2s	620s	6.2s	62s
24hr	21666	3 per s	6.2s	620s	6.2s	744s

The *Clean* algorithm, below, is used periodically 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 ← ⋂_{x_j ∈ PrizeSet} x_j.a                           ▷ O(n(n + m))
  converged ← converged ∩ DepletedSet                            ▷ O(n + m)
  for x_j ∈ converged do
      DepletedSet ← DepletedSet\{x_j}                            ▷ O(log(m))
      ConvergedDepletedSet ← ConvergedDepletedSet ∪ {x_j}        ▷ O(1)
  end for
  for x_j ∈ PrizeSet do
      x_j.a ← x_j.a\converged                                    ▷ O(n(n + m))
  end for
```

D SIMULATOR AND EXPERIMENTS

We have implemented a simulator to check throughput, and ascertain experimentally that the proposed algorithms induce the right system behaviour. Our implementation is written in Rust. It includes a wallet and a simulation suite.

One interesting observation is that the whole system converges very quickly even if their parents are chosen randomly.

Table 1 collects statistics from the simulations. Of particularly interest are the last three rows (data rows 6, 7 and 8) where the simulation has been tuned to have an equivalent transaction rate as the current Bitcoin network, adjusted for simulation length in order to achieve the same median network delay. [3] The upshot is that our simulation, running on a single i7-4770 CPU (with no optimization) was easily able to match the transaction rate of the current Bitcoin network and verify all transactions in real time.

ACKNOWLEDGEMENTS

The first author is supported by an Australian Research Council Future Fellowship under grant no. FT140101145.

[3] Using Bitcoin's peak rate of 260,000 tx/day on 2016/11/07 per `http://www.coindesk.com/data/bitcoin-daily-transactions`, matched to the median propagation delay on 2016/03/07 per `http://bitcoinstats.com/network/propagation`.

REFERENCES

[1] Simon Barber, Xavier Boyen, Elaine Shi, and Ersin Uzun. 2012. Bitter to Better - How to Make Bitcoin a Better Currency. In *FC 2012 (LNCS)*, Angelos D. Keromytis (Ed.), Vol. 7397. Springer, Heidelberg, 399–414.

[2] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. 2014. Zerocash: Decentralized Anonymous Payments from Bitcoin, 459–474. https://doi.org/10.1109/SP.2014.36

[3] Iddo Bentov, Pavel Hubáček, Tal Moran, and Asaf Nadler. 2017. Tortoise and Hares Consensus: the Meshcash Framework for Incentive-Compatible, Scalable Cryptocurrencies. http://eprint.iacr.org/2017/300. (2017).

[4] Joseph Bonneau, Andrew Miller, Jeremy Clark, Arvind Narayanan, Joshua A. Kroll, and Edward W. Felten. 2015. SoK: Research Perspectives and Challenges for Bitcoin and Cryptocurrencies. In *2015 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 104–121. https://doi.org/10.1109/SP.2015.14

[5] Xavier Boyen, Christopher Carr, and Thomas Haines. 2016. Blockchain-Free Cryptocurrencies: A Framework for Truly Decentralised Fast Transactions. https://eprint.iacr.org/2016/871. (2016).

[6] Miles Carlsten, Harry A. Kalodner, S. Matthew Weinberg, and Arvind Narayanan. 2016. On the Instability of Bitcoin Without the Block Reward. In *ACM CCS 16*, Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi (Eds.). ACM Press, 154–167.

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

[8] David Chaum, Amos Fiat, and Moni Naor. 1990. Untraceable Electronic Cash. In *CRYPTO'88 (LNCS)*, Shafi Goldwasser (Ed.), Vol. 403. Springer, Heidelberg, 319–327.

[9] George Danezis and Sarah Meiklejohn. 2016. Centrally Banked Cryptocurrencies. In *NDSS 2016*. The Internet Society.

[10] Christian Decker and Roger Wattenhofer. [n. d.]. A Fast and Scalable Payment Network with Bitcoin Duplex Micropayment Channels. In *Stabilization, Safety, and Security of Distributed Systems - 17th International Symposium, SSS 2015, Edmonton, AB, Canada, August 18-21, 2015, Proceedings* Andrzej Pelc and Alexander A. Schwarzmann (Eds.), Vol. 9212. Springer.

[11] Christian Decker and Roger Wattenhofer. [n. d.]. Information propagation in the Bitcoin network. In *13th IEEE International Conference on Peer-to-Peer Computing, IEEE P2P 2013, Trento, Italy, September 9-11, 2013, Proceedings*. IEEE.

[12] Cynthia Dwork and Moni Naor. 1993. Pricing via Processing or Combatting Junk Mail. In *CRYPTO'92 (LNCS)*, Ernest F. Brickell (Ed.), Vol. 740. Springer, Heidelberg, 139–147.

[13] Ethan Heilman and Neha Narula and Thaddeus Dryja and Madars Virza. 2017. IOTA Vulnerability Report: Cryptanalysis of the Curl Hash Function Enabling Practical Signature Forgery Attacks on the IOTA Cryptocurrency. https://github.com/mit-dci/tangled-curl/blob/master/vuln-iota.md. (2017). [Acc: Jan'18].

[14] Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert van Renesse. [n. d.]. Bitcoin-NG: A Scalable Blockchain Protocol. In *13th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2016, Santa Clara, CA, USA, March 16-18, 2016*, Katerina J. Argyraki and Rebecca Isaacs (Eds.). USENIX Association.

[15] Ittay Eyal and Emin Gün Sirer. 2014. Majority Is Not Enough: Bitcoin Mining Is Vulnerable. In *FC 2014 (LNCS)*, Nicolas Christin and Reihaneh Safavi-Naini (Eds.), Vol. 8437. Springer, Heidelberg, 436–454. https://doi.org/10.1007/978-3-662-45472-5_28

[16] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. 2015. The Bitcoin Backbone Protocol: Analysis and Applications. In *EUROCRYPT 2015, Part II (LNCS)*, Elisabeth Oswald and Marc Fischlin (Eds.), Vol. 9057. Springer, Heidelberg, 281–310. https://doi.org/10.1007/978-3-662-46803-6_10

[17] Arthur Gervais, Ghassan O. Karame, Vedran Capkun, and Srdjan Capkun. 2014. Is Bitcoin a Decentralized Currency? *IEEE Security & Privacy* 12, 3 (2014), 54–60.

[18] Benjamin Johnson, Aron Laszka, Jens Grossklags, Marie Vasek, and Tyler Moore. 2014. Game-Theoretic Analysis of DDoS Attacks Against Bitcoin Mining Pools. In *FC 2014 Workshops (LNCS)*, Rainer Böhme, Michael Brenner, Tyler Moore, and Matthew Smith (Eds.), Vol. 8438. Springer, Heidelberg, 72–86. https://doi.org/10.1007/978-3-662-44774-1_6

[19] Ari Juels and John G. Brainard. 1999. Client Puzzles: A Cryptographic Countermeasure Against Connection Depletion Attacks. In *NDSS'99*. The Internet Society.

[20] Ghassan Karame, Elli Androulaki, and Srdjan Capkun. 2012. Double-spending fast payments in bitcoin. In *ACM CCS 12*, Ting Yu, George Danezis, and Virgil D. Gligor (Eds.). ACM Press, 906–917.

[21] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. [n. d.]. Enhancing Bitcoin Security and Performance with Strong Consistency via Collective Signing. In *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.*, Thorsten Holz and Stefan Savage (Eds.). USENIX Association.

[22] Sergio Demian Lerner. 2015. DagCoin Draft. https://bitslog.files.wordpress.com/2015/09/dagcoin-v41.pdf. (2015). [Acc: Jan'18].

[23] Yoad Lewenberg, Yonatan Sompolinsky, and Aviv Zohar. 2015. Inclusive Block Chain Protocols. In *FC 2015 (LNCS)*, Rainer Böhme and Tatsuaki Okamoto (Eds.), Vol. 8975. Springer, Heidelberg, 528–547. https://doi.org/10.1007/978-3-662-47854-7_33

[24] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M. Voelker, and Stefan Savage. 2016. A fistful of Bitcoins: characterizing payments among men with no names. *Commun. ACM* 59, 4 (2016).

[25] Ian Miers, Christina Garman, Matthew Green, and Aviel D. Rubin. 2013. Zerocoin: Anonymous Distributed E-Cash from Bitcoin. In *2013 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 397–411.

[26] Andrew Miller, Ari Juels, Elaine Shi, Bryan Parno, and Jonathan Katz. 2014. Permacoin: Repurposing Bitcoin Work for Data Preservation, 475–490. https://doi.org/10.1109/SP.2014.37

[27] Andrew Miller, Ahmed E. Kosba, Jonathan Katz, and Elaine Shi. 2015. Nonoutsourceable Scratch-Off Puzzles to Discourage Bitcoin Mining Coalitions. In *ACM CCS 15*, Indrajit Ray, Ninghui Li, and Christopher Kruegel: (Eds.). ACM Press, 680–691.

[28] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. (2008). Available at https://bitcoin.org/bitcoin.pdf.

[29] Arvind Narayanan, Joseph Bonneau, Edward W. Felten, Andrew Miller, and Steven Goldfeder. 2016. *Bitcoin and Cryptocurrency Technologies - A Comprehensive Introduction*. Princeton University Press. ISBN: 978-0-691-17169-2.

[30] Ethereum Network. 2013. Ethereum: Smart Contract and Decentralized Application Platform. https://github.com/ethereum/wiki/wiki/White-Paper. (2013). [Acc: Jan'18].

[31] Sunoo Park, Krzysztof Pietrzak, Albert Kwon, Joël Alwen, Georg Fuchsbauer, and Peter Gaži. 2015. Spacemint: A Cryptocurrency Based on Proofs of Space. Cryptology ePrint Archive, Report 2015/528. (2015). http://eprint.iacr.org/2015/528.

[32] Rafael Pass, Lior Seeman, and Abhi Shelat. 2017. Analysis of the Blockchain Protocol in Asynchronous Networks. In *EUROCRYPT 2017, Part II (LNCS)*, Jean-Sébastien Coron and Jesper Buus Nielsen (Eds.), Vol. 10211. Springer, 643–673.

[33] Joseph Poon and Thaddeus Dryja. 2016. The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments. https://lightning.network/lightning-network-paper.pdf. (2016). [Acc: Jan'18].

[34] Serguei Popov. 2017. IOTA: The tangle. http://iotatoken.com/IOTA_Whitepaper.pdf. (2017). [Acc: Jan'18].

[35] David Schwartz, Noah Youngs, and Arthur Britto. 2014. The Ripple protocol consensus algorithm. (2014). [Acc: Jan'18].

[36] Yonatan Sompolinsky, Yoad Lewenberg, and Aviv Zohar. 2016. SPECTRE: A Fast and Scalable Cryptocurrency Protocol. Cryptology ePrint Archive, Report 2016/1159. (2016). http://eprint.iacr.org/2016/1159.

[37] Yonatan Sompolinsky and Aviv Zohar. 2013. Accelerating Bitcoin's Transaction Processing. Fast Money Grows on Trees, Not Chains. Cryptology ePrint Archive, Report 2013/881. (2013). http://eprint.iacr.org/2013/881.

[38] Douglas Stebila, Lakshmi Kuppusamy, Jothi Rangasamy, Colin Boyd, and Juan Manuel González Nieto. 2011. Stronger Difficulty Notions for Client Puzzles and Denial-of-Service-Resistant Protocols. In *CT-RSA 2011 (LNCS)*, Aggelos Kiayias (Ed.), Vol. 6558. Springer, Heidelberg, 284–301.

[39] Florian Tschorsch and Björn Scheuermann. 2016. Bitcoin and Beyond: A Technical Survey on Decentralized Digital Currencies. *IEEE Communications Surveys and Tutorials* 18, 3 (2016), 2084–2123.

[40] Web. 2011. Litecoin. https://litecoin.org/ (2011). [Acc: Jan'18].

[41] Web. 2013. Dogecoin. http://dogecoin.com/ (2013). [Acc: Jan'18].

[42] Web. 2014. Dan Goodin Ars Technica. http://arstechnica.com/security/2014/06/bitcoin-security-guarantee-shattered-by-anonymous-miner-with-51-network-power (2014). [Acc: Jan'18].
[43] Web. 2014. Stellar. https://www.stellar.org/ (2014). [Acc: Jan'18].
[44] Web. 2016. IOTA. http://iota.org/ (2016). [Acc: Jan'18].
[45] Web. 2017. Hashgraph: The future of decentralised technology. https://hashgraph.com (2017). [Acc: Jan'18].