International Conference on Blockchain Economics, Security and Protocols

Tokenomics 2019, May 6–7, 2019, Paris, France

Edited by

Vincent Danos
Maurice Herlihy
Maria Potop-Butucaru
Julien Prat
Sara Tucci-Piergiovanni

Editors

Vincent Danos
ENS, CNRS, PSL University, Paris, France
INRIA, Paris, France
Vincent.Danos@ens.fr

Maurice Herlihy
Brown University, Providence, RI, USA
mph@cs.brown.edu

Maria Potop-Butucaru
Sorbonne Université, Paris, France
maria.potop-butucaru@lip6.fr

Julien Prat
CREST, Ecole Polytechnique, Palaiseau, France
Julien.Prat@ensae.fr

Sara Tucci-Piergiovanni
CEA LIST, Palaiseau, France
sara.tucci@cea.fr

ACM Classification 2012
Computing methodologies → Distributed algorithms; Security and privacy → Distributed systems security; Applied computing → Economics

ISBN 978-3-95977-108-5

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at https://www.dagstuhl.de/dagpub/978-3-95977-108-5.

Publication date
March, 2020

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at https://portal.dnb.de.

Digital Object Identifier: 10.4230/OASIcs.Tokenomics.2019.0
ISBN 978-3-95977-108-5 **ISSN 1868-8969** **https://www.dagstuhl.de/oasics**

OASIcs – OpenAccess Series in Informatics

OASIcs aims at a suitable publication venue to publish peer-reviewed collections of papers emerging from a scientific event. OASIcs volumes are published according to the principle of Open Access, i.e., they are available online and free of charge.

ISSN 1868-8969

https://www.dagstuhl.de/oasics

Contents

International Conference on Blockchain Economics, Security and Protocols (Tokenomics 2019).
Editors: Vincent Danos, Maurice Herlihy, Maria Potop-Butucaru, Julien Prat, and Sara Tucci-Piergiovanni
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Preface

Blockchains enable new trust architectures. These architectures are not just distributed but they are also *decentralised*. This means that no single person or entity is specifically in charge of running the system – no one runs the show. Instead, the system is arranged in a way that many *unrelated* agents are collectively in charge. From their collective effort emerges ideally a perfect and transparent trusted third party. It is not easy to organise the production of such a decentralised resource. It is the work of the blockchain engine – or consensus algorithm – to keep the many operators with consistent views of the system and to make the system reliable and dependable. Systems built in this way offer no specific point of failure – so the story goes. For the same reason, they are hard and costly to attack, as the adversary needs to recruit massive resources to coordinate sufficiently many agents to take over the system.

Blockchains have been around for some time now and continue to grow in use and in diversity of services and underpinning mechanisms. There are many trust engines – each with different trade-offs between various criteria of performance and various levels of maturity. The creativity in the field is truly staggering and a consequence of its openness. Note that openness is a natural correlate of decentralisation: if the system is closed, it is the beginning of an inventory of the various agents and one eventually will come to know who they are. People love openness as they have direct access to the levers of governance/consensus and monetary policies, things most people never get to see in a lifetime except when playing video games maybe. This combination of openness and of having a computational substrate opens up new algorithmic economic spaces. Voting, allocation, rewards, monetary policies, pricing cascades, tax systems everything is up for reinvention in a framework where everything can be seen and everyone sees that rules are correctly applied. In this enthusiastic universe we have already seen crises, collapses, and subsequent evolution. We have now some data points and an incipient understanding of what works and what not.

However, there are many things the field still needs to improve. One which is most frequently mentioned is the need to *scale up* decentralised systems to a level comparable to that of the traditional trust systems they purport to replace (at least in part). Another one is the ability to offer *confidentiality-preserving* modes of operation as the need of secrecy is often a necessity in business transactions. Also it would help if the price of the collective computational resource was lower (compared to centralised cloud computing) and more predictable.

But there are other pressing and perhaps more difficult questions. No matter what technique is used for consensus, it all relies on a key assumption: namely that the multiple agents in charge are *independent* or approximately so - and will therefore act neutrally with respect to the users and be only driven by their own interests. Agents do not collude - it is assumed. The trust therefore derives not from traditional *reputation-and-regulation* mechanisms but from this neutrality postulate on which everything blockchain hinges. For this assumption to be realistic, one needs many agents - so many that no actor can summon enough resources to corrupt or otherwise control a sufficiently large subset of the agents of the system.

It follows that such systems are harder to update and to set back on a good course should there be a problem. It is a logical necessity that the system has no single point of accountability. It is also hard to repair or amend a system with many independent operators. One needs means of coordination and yet no means of collusion. It sounds difficult and it is!

International Conference on Blockchain Economics, Security and Protocols (Tokenomics 2019).
Editors: Vincent Danos, Maurice Herlihy, Maria Potop-Butucaru, Julien Prat, and Sara Tucci-Piergiovanni
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

One fundamental need of the economic world may be a need for some level of reversibility that is hard to combine with decentralised mechanisms. Another aspect of the coordination-without-collusion problem is that there is not even the common legally operative notion of a "one" agent - meaning the various software agents (commonly called nodes, miners, block makers, and users) that maintain and provide the computational resources of the system are not legal entities (persons or other type of legal entity such as firms). This is another fundamental and fundamentally unanswered question at the time of writing. Namely, how one can even define and measure – let alone incentivise – decentralisation. Counting how many nodes there are can only be a proxy. This is one aspect of the famous Sybil problem: namely the near-zero cost of creating new on-line pseudo-identities. Yet another unavoidable correlate of decentralisation is that agents in the system need to find an incentive to maintain that system - and that incentive has to be baked in the decentralised operation (else the nodes are someone's employee or friends and the system is no longer de-centralised).

The ambition of the conference we have organised – which we hope will be the first of a series – was specifically to cater to this broad type of questions around the incentive structures that are needed to keep up and stabilize decentralised systems. How does one measure, induce, and monitor "decentralised" in a decentralised environment. If trust is a resource, how much trust does one need for what usage and at what price. How does one design incentives that will hold the trust-providing system together, keep its different actors happy, and foster stability and resilience. Specifically, how does one set up the rules for the allocation of platform profits, and how does one handle the profit/price dilemma. That is to say how can one reconcile profit distribution rules with the price of the system's own token/cryptocurrency which is the means by which incentives are implemented. Token holders want a high price but platform contributors want a high profit.

There is need for methodological guidance to find both pen and paper and data-driven solution paths to the key questions above; to produce tools that will help designers of decentralised socio-technical systems to 'science out' fundamental difficulties; to reinforce and build better trust structures with sound economics; and understand the complex multi-objective optimisation which is implied. We hope this conference and its subsequent editions will provide a favourable space for the further exploration of these new territories in computer science and economics.

For this first edition, there were 38 papers submitted: 23 papers in computer science (17 as regular papers for publication in the proceedings and 6 for presentation only) and 16 papers in economics. The computer science program committee selected 11 papers for publication and presentation and 3 papers for presentation only. The economics program committee selected 8 papers for presentation at the conference. Every submitted paper was evaluated by at least three members of the program committee.

The program included two keynotes lecture in computer science by Amr El Abbadi (UCSB, USA) and Dahlia Malkhi (VMware, USA) and two keynotes lectures in economics by Lin William Cong (University of Chicago Booth School of Business, USA) and Catherine Casamatta (Toulouse School of Economics, France). The abstracts of the keynote lectures in computer science are included in this volume.

The best paper award was presented by William George and Clement Leseage for the paper:*a Smart Contract Oracle for Approximating Real-World, Real Number Values*. The paper was awarded with the "Asseth - Kaiko Prize for Research in Cryptoeconomics" (1,500 euros).

The success of this first edition was the result of a team effort. We thank the authors for providing valuable content for the conference and the program committee who worked hard in reviewing papers and giving feedback to the authors. We also thank the Ecole

Normale Superieure who hosted the conference, our institutional supporters, CEA LIST, CNRS, CREST, ENS, Sorbonne Université, and our financial supporters, Asseth-Kaiko, Capgemini, Institut Carnot. Finally, we want to thank our wonderful students and colleagues, Antonella, Yackolley, Pablo, Gewu, Nicolas, Zeinab, Zainah, Agnès, Onder, Thibault, Aimen and Francois that helped with all the logistics of the conference.

Maria, Vincent, Sara, Julien and Maurice

Tokenomics 2019 Organization

General Chairs

Vincent Danos, Ecole Normale Supérieure (France)
Maurice Herlihy, Brown University (USA)
Maria Potop-Butucaru, Sorbonne Université (France)
Julien Prat, CREST, Ecole Polytechnique (France)
Sara Tucci-Piergiovanni, CEA LIST (France)

Program Committee

Computer Science

Emmanuelle Anceaume, IRISA (France)
Antonio Fernández Anta, IMDEA Networks (Spain)
Daniel Augot, INRIA, Ecole Polytechnique (France)
Konstantinos Chalkias, R3 (UK, USA)
Vincent Danos, Ecole Normale Supérieure (France)
Fabrice Le Fessant, OCaml PRO (France)
Bryan Ford, Ecole Polytechnique Fédérale de Lausanne - EPFL (Switzerland)
Georg Fuchsbauer, Ecole Normale Supérieure (France)
Juan A. Garay. Texas A&M University (USA)
Chryssis Georgiou, University of Cyprus (Cyprus)
Vincent Gramoli, The University of Sydney (Australia)
Rachid Guerraoui, Ecole Polytechnique Fédérale de Lausanne - EPFL (Switzerland)
Maurice Herlihy, Brown University (USA)
Guillaume Jourjon, Data61-CSIRO (Australia)
Aggelos Kiayias, The University of Edinburgh (UK)
Mario Larangeira, IOHK, Tokyo Institute of Technology (Japan)
Žarko Milošević. Tendermint, Singidunum University (Serbia)
Maria Potop-Butucaru, Sorbonne Université (France)
Michel Raynal, IRISA-IFSIC, Institut Universitaire de France (France)
Ilya Sergey, Yale-NUS College (Singapore)
Alexander Spiegelman, VMware Research Group (Israel)
Francois Taiani, INRIA, IRISA (France)
Sara Tucci-Piergiovanni, CEA LIST (France)
Marko Vukolic, IBM Research - Zurich (Switzerland)
Roger Wattenhofer, ETH Zurich (Switzerland)
Josef Widder, TU Wien (Austria)

Economics

Bruno Biais, HEC (France)
Christophe Bisière, Toulouse School of Economics (France)
Andrea Canidio, IMT and INSEAD (Italy)
Lin William Cong, University of Chicago Booth School of Business (USA)
Guillaume Haeringer, Baruch College (USA)

International Conference on Blockchain Economics, Security and Protocols (Tokenomics 2019).
Editors: Vincent Danos, Maurice Herlihy, Maria Potop-Butucaru, Julien Prat, and Sara Tucci-Piergiovanni
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Hanna Halaburda, NYU Stern School of Business (USA)
Gur Huberman, Columbia University (USA)
Winfried Koeniger, St-Gallen University (Switzerland)
Gina Pieters, University of Chicago (USA)
Julien Prat, CREST, Ecole Polytechnique (France)
Linda Schilling, Ecole Polytechnique (France)

Demystifying Blockchains: Decentralized and Fault-Tolerant Storage for the Future of Big Data?

Amr El Abbadi[1]
University of California of Santa Barbara, USA

Abstract

Bitcoin is a successful and interesting example of a global scale peer-to-peer cryptocurrency that integrates many techniques and protocols from cryptography, distributed systems, and databases. The main underlying data structure is blockchain, a scalable fully replicated structure that is shared among all participants and guarantees a consistent view of all user transactions by all participants in the cryptocurrency system. The novel aspect of Blockchain is that historical data about all transactions is maintained in the absence of any central authority. This property of Blockchain has given rise to the possibility that future applications will transition from centralized databases to a fully decentralized storage based on blockchains. In this talk, we start by developing an understanding of the basic protocols used in blockchain, and elaborate on their main advantages and limitations. To overcome these limitations, we will explore some of the challenges of managing large scale fully replicated ledgers in the context of achieving large scale consensus. Finally, we ponder over recent efforts to use blockchains in diverse applications.

2012 ACM Subject Classification Computing methodologies → Distributed algorithms

Keywords and phrases distributed algorithms for databases, distributed storage, blockchains

Digital Object Identifier 10.4230/OASIcs.Tokenomics.2019.1

Category Keynote Lecture

Bio. Amr El Abbadi is a Professor of Computer Science at the University of California, Santa Barbara. He received his B. Eng. from Alexandria University, Egypt, and his Ph.D. from Cornell University. Prof. El Abbadi is an ACM Fellow, AAAS Fellow, and IEEE Fellow. He was Chair of the Computer Science Department at UCSB from 2007 to 2011. He has served as a journal editor for several database journals, including, The VLDB Journal, IEEE Transactions on Computers and The Computer Journal. He has been Program Chair for multiple database and distributed systems conferences. He currently serves on the executive committee of the IEEE Technical Committee on Data Engineering (TCDE) and was a board member of the VLDB Endowment from 2002 to 2008. In 2007, Prof. El Abbadi received the UCSB Senate Outstanding Mentorship Award for his excellence in mentoring graduate students. In 2013, his student, Sudipto Das received the SIGMOD Jim Gray Doctoral Dissertation Award. Prof. El Abbadi is also a co-recipient of the Test of Time Award at EDBT/ICDT 2015. He has published over 300 articles in databases and distributed systems and has supervised over 35 PhD students.

[1] in collaboration with: Divy Agrawal, Mohammad Amiri, Sujaya Maiyya, Victor Zakhary

International Conference on Blockchain Economics, Security and Protocols (Tokenomics 2019).
Editors: Vincent Danos, Maurice Herlihy, Maria Potop-Butucaru, Julien Prat, and Sara Tucci-Piergiovanni;
Article No. 1; pp. 1:1–1:1

OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Flexible BFT: Separating BFT Protocol Design from the Fault Model

Dahlia Malkhi
VMWare
https://dahliamalkhi.wordpress.com/

Abstract

Byzantine Fault Tolerant (BFT) protocols designed for building replicated services collapse if deployed under settings that differ from the fault model they are designed for. For example, in a partial-synchrony model, a known lower bound for BFT is 1/3. Optimal-resilience solutions completely break if the fraction of Byzantine faults exceeds 1/3. The only way we know to achieve $> 1/3$ resilience is by assuming synchrony, but this requires the protocol to be designed with that assumption. Flexible BFT is a new approach to BFT protocol design that separates between the fault model and the solution. Clients in Flexible BFT specify (i) the adversarial threshold they need to tolerate, and (ii) whether they believe in synchrony (and the presumed bound on transmission delays). We present a Flexible BFT solution that simultaneously supports different clients, who differ simply by the number of messages and/or time the clients are willing to wait for. At an even finer grain, Flexible BFT supports under the same solution high-value and low-value transactions, each tolerating a different threat model.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Byzantine fault-tolerance, blockchains

Digital Object Identifier 10.4230/OASIcs.Tokenomics.2019.2

Category Keynote Lecture

Bio. Dahlia Malkhi carries applied and foundation research in broad aspects of reliability and security in distributed systems since the early nineties. In 2014, after the closing of the Microsoft Research Silicon Valley lab, she co-founded VMware Research and became a Principal Researcher at VMware. From 2004-2014, she was a principal researcher at Microsoft Research, Silicon Valley. From 1999-2007, she was a tenured associate professor at the Hebrew University of Jerusalem. In 2004, leaving for a brief sabbatical at Microsoft Research, she was bitten by the Silicon Valley bug and stayed there. Dr. Malkhi was elected ACM fellow in 2011, received the IBM Faculty award in 2003 and 2004, and the German-Israeli Foundation (G.I.F.) Young Scientist career award 2002. She currently co-leads the VMware blockchain research project. In the past decade, she founded and led the Corfu project, a database-less database. The Corfu data platform currently drives VMware's NSX-T distributed control plane.

International Conference on Blockchain Economics, Security and Protocols (Tokenomics 2019).
Editors: Vincent Danos, Maurice Herlihy, Maria Potop-Butucaru, Julien Prat, and Sara Tucci-Piergiovanni;
Article No. 2; pp. 2:1–2:1

OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

A Puff of Steem: Security Analysis of Decentralized Content Curation

Aggelos Kiayias
University of Edinburgh, United Kingdom
IOHK, Hong Kong

Benjamin Livshits
Imperial College of London, United Kingdom
Brave Software, United Kingdom

Andrés Monteoliva Mosteiro
University of Edinburgh, United Kingdom
Clearmatics, London, United Kingdom

Orfeas Stefanos Thyfronitis Litos[1]
University of Edinburgh, United Kingdom
o.thyfronitis@ed.ac.uk

Abstract

Decentralized content curation is the process through which uploaded posts are ranked and filtered based exclusively on users' feedback. Platforms such as the blockchain-based Steemit[2] employ this type of curation while providing monetary incentives to promote the visibility of high quality posts according to the perception of the participants. Despite the wide adoption of the platform very little is known regarding its performance and resilience characteristics. In this work, we provide a formal model for decentralized content curation that identifies salient complexity and game-theoretic measures of performance and resilience to selfish participants. Armed with our model, we provide a first analysis of Steemit identifying the conditions under which the system can be expected to correctly converge to curation while we demonstrate its susceptibility to selfish participant behaviour. We validate our theoretical results with system simulations in various scenarios.

2012 ACM Subject Classification Security and privacy → Distributed systems security

Keywords and phrases blockchain, content curation, decentralized, voting

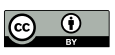
Digital Object Identifier 10.4230/OASIcs.Tokenomics.2019.3

Related Version A full version of the paper is available at `https://arxiv.org/abs/1810.01719`.

1 Introduction

The modern Internet contains an immense amount of data; a single user can only consume a tiny fraction in a reasonable amount of time. Therefore, any widely used platform that hosts user-generated content (UGC) must employ a content curation mechanism. Content curation can be understood as the set of mechanisms which rank, aggregate and filter relevant information. In recent years, popular news aggregation sites like Reddit[3] or Hacker News[4] have established crowdsourced curation as the primary way to filter content for their users. Crowdsourced content curation, as opposed to more traditional techniques such as expert- or

[1] Contact author
[2] `https://steemit.com/` Accessed: 2019-01-02
[3] `https://www.reddit.com/` Accessed: 2019-01-02
[4] `https://news.ycombinator.com/` Accessed: 2019-01-02



International Conference on Blockchain Economics, Security and Protocols (Tokenomics 2019).
Editors: Vincent Danos, Maurice Herlihy, Maria Potop-Butucaru, Julien Prat, and Sara Tucci-Piergiovanni;
Article No. 3; pp. 3:1–3:21

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

algorithmic-based curation, orders and filters content based on the ratings and feedback of the users themselves, obviating the need for a central moderator by leveraging the "wisdom of the crowd" [3, 46].

The decentralized nature of crowdsourced curation makes it a suitable solution for ranking user-generated content in blockchain-based content hosting systems. The aggregation and filtering of user-generated content emerges as a particularly challenging problem in permissionless blockchains, as any solution that requires a concrete moderator implies that there exists a privileged party, which is incompatible with a permissionless blockchain. Moreover, public blockchains are easy targets for Sybil attacks [10], as any user can create new accounts at any time for a marginal cost. Therefore, on-chain mechanisms to resist the effect of Sybil users are necessary for a healthy and well-functioning platform; traditional counter-Sybil mechanisms [29] are much harder to apply in the case of blockchains due to the decentralized nature of the latter. The functions performed by moderators in traditional content platforms need to be replaced by incentive mechanisms that ensure self-regulation. Having the impact of a vote depend on the number of coins the voter holds is an intuitively appealing strategy to achieve a proper alignment of incentives for users in decentralized content platforms; specifically, it can render Sybil attacks impossible.

However, the correct design of such systems is still an unsolved problem. Blockchains have created a new economic paradigm where users are at the same time equity holders in the system, and leveraging this property in a robust manner constitutes an interesting challenge. A variety of projects have designed decentralized content curation systems [27, 42, 16]. Nevertheless, a deep understanding of the properties of such systems is still lacking. Among them, Steemit has a long track record, having been in operation since 2016 and attaining a user base of more than 1.08 M[5] registered accounts[6]. Steemit is a social media platform which lets users earn money (in the form of the STEEM cryptocurrency) by both creating and curating content in the network. Steemit is the front-end of the social network, a graphical web interface which allows users to see the content of the platform. On the other hand, all the back-end information is stored on a distributed ledger, the Steem blockchain. Steem can be understood as an "app-chain", a blockchain with a specific application purpose: serving as a distributed database for social media applications [42].

1.1 Our Contributions

In this work we study the foundations of decentralized content curation from a computational perspective. We develop an abstract model of a post-voting system which aims to sort the posts created by users in a distributed and crowdsourced manner. Our model is constituted by a functionality which executes a protocol performed by N players. The model includes an honest participant behaviour while it allows deviations to be modeled for a subset of the participants. The N players contribute votes in a round-based curation process. The impact of each vote depends on the number of coins held by the player. The posts are arranged in a list, sorted by the value of votes received, resembling the front-page model of Reddit or Hacker News. In the model, players vote according to their subjective opinion on the quality of the posts and have a limited attention span.

Following previous related work [14, 3], we represent each player's opinion on each post (i.e. likability) with a numerical value $l \in [0, 1]$. The objective quality of a post is calculated as the simple summation of all players' likabilities for the post in question. To measure

[5] `https://steemdb.com/accounts` Accessed: 2019-01-02

[6] The number of accounts should not be understood as the number of active users, as one user can create multiple accounts.

the effectiveness of a post-voting system, we introduce the property of *convergence* under honesty which is parameterised by a number of values including a metric t, that demands the first t articles to be ordered according to the objective quality of the posts at the end of the execution assuming all participants signal honestly to the system their personal preferences. Armed with our post-voting system abstraction, we proceed to particularize it to model Steemit and provide the following results.

(i) We characterise the conditions under which the Steemit algorithm converges under honesty. Our results highlight some fundamental limitations of the actual Steemit parameterization. Specifically, for curated lists of length bigger than 70 the algorithm may *not achieve even 1-convergence*.

(ii) We validate our results with a simulation testing different metrics based on correlation that have been proposed in previous works [25, 37] and relating them to our notion of convergence.

(iii) We demonstrate that "selfish" deviation from honest behavior results to substantial gains in terms of boosting the ranking of specific posts in the resulting list of the post-voting system, and to a grave reduction of the quality of said list.

1.2 Steem consensus algorithm

In a nutshell, Delegated Proof of Stake [8, 36, 41] works as follows: Steem users can sign up as "validator" candidates for one of 21 slots. Each user that owns some STEEM can vote for a validator. The 20 candidates that receive the most votes (weighted by the respective users' STEEM) become validators. The 21st slot is filled with one of the candidates that was not elected, chosen at random with probability proportional to her votes.

A validator is responsible for receiving new transactions and adding them to blocks. Validators take turns in block production. An honest validator attaches her block to the latest valid block she knows and broadcasts it to the network. We say that a round is complete after each validator has had a chance to create a block. Honest nodes accept the longest known chain as the valid one. Elections for validators happen once each round, thus each STEEM holder is allowed to change her opinion very often.

The protocol promises that all new transactions are permanently added to the blockchain in a short amount of time, given that at least two thirds of the validators are honest. Unfortunately, we were unable to locate a formal proof of this claim.

Note that our analysis does not focus on DPOS, but on the curation mechanism of Steemit. The latter is independent of the consensus protocol of Steem.

2 Related Work

User-generated content (UGC) has been identified as a fundamental component of social media platforms and Web 2.0 in general [24]. The content created by users needs to be curated, and crowdsourced content curation [3] has emerged as an alternative to expert-based [38] or algorithmic-based [35] curation techniques. Motivated by the widespread adoption of crowdsourced aggregation sites such as Reddit or Digg[7], several research efforts [9, 14, 1] have aimed to model the mechanics and incentives for users in UGC platforms. This surge of interest is accompanied by studies which have shown how social media users behave strategically when they publish and consume content [32]. As an example, in the case of Reddit, users try to maximize their "karma" [4], the social badge of the social media platform [2].

[7] `http://digg.com/` Accessed: 2019-01-02

Previous works have analyzed content curation from an incentives and game-theoretic standpoint [14, 9, 21, 32, 1] . Our formalisation is based on these models and inherits features such as the quality distribution of the articles and the users' attention span [3, 14]. In terms of the analysis of our results, the analysis of our *t-convergence* metric is similar to the top-k posts in [3]. We also leverage the rank correlation coefficients Kendall's Tau [25] and Spearman's Rho [37] to measure content curation efficiency. Our approach describes the mechanics of post-voting systems from a computational perspective, something that departs from the approach of all previous works, drawing inspiration from the real-ideal world paradigm of cryptography [17, 30] as employed in our definition of t-convergence.

Post-voting systems constitute a special case of voting mechanisms, as studied within social choice theory, belonging to the subcategory of cardinal voting systems [22]. In this context, it follows from Gibbard's theorem [15] that no decentralised non-trivial post-voting mechanism can be strategy-proof. This is consistent with our results that demonstrate how selfish behaviour is beneficial to the participants. Our system shares the property of spanning multiple voting rounds with previous work [23]. Other related literature in social choice [31, 6, 44] is centered on political elections and as a result attempts to resolve a variation of the problem with quite different constraints and assumptions. In more detail, in the case of political elections, voter communication in many rounds is costly while navigating the ballot is not subject to any constraints as voters are assumed to have plenty of time to parse all the options available to them. As a result, voters can express their preferences for any candidate, irrespective of the order in which the latter appear on the ballot paper. On the other hand, the online and interactive nature of post-voting systems make multi-round voting a natural feature to be taken advantage of. At the same time, the fairness requirements are more lax and it is acceptable (even desirable) for participants to act reactively on the outcome of each others' evaluations. On the other hand, in the post-voting case, the "ballot" is only partially available given the high number of posts to be ranked that may very well exceed the time available to a (human) user to participate in the process. As a result a user will be unable to vote for posts that she has not viewed, for instance, because they are placed at the bottom of the list. This is captured in our model by introducing the concept of "attention span".

Content curation is also related to the concept of online governance. The governance of online communities such as Wikipedia has been thoroughly studied in previous academic work [28, 13]. However, the financially incentivized governance processes in blockchain systems, where the voters are at the same time equity-holders, have still many open research questions [5, 12]. This shared ownership property has triggered interest in building social media platforms backed by distributed ledgers, where users are rewarded for generated content and variants of coin-holder voting are used to decide how these rewards are distributed.

As already mentioned, coin-weighted voting is a viable mechanism to measure the influence of users in the platform and, by extension, to make the system more resistant to Sybil attacks. Different countermeasures for the Sybil problem in content curation and recommendation sites have been explored in the past [34, 40, 45, 33]. Orthogonal to the coin-weighted voting model, these solutions leverage the trust graph of the underlying social network (which is explicitly created by users) to bound the effect of Sybil votes [34, 40, 45]. [43] claim that trust graph-based solutions require heavy computation, and propose optimizations for real-world applications modeling the transitive trust relationships as credit networks. We acknowledge these mechanisms as complementary to coin-weighted voting and potentially implementable in Steemit. We note that the abstract post-voting system defined in this work can be particularized to include such trust graph-based solutions.

The effects of explicit financial incentives on the quality of content in Steemit has been analyzed in [39]. Beyond the Steemit's whitepaper [42], a series of blog posts [18, 19] effectively extend the economic analysis of the system. In parallel with Steemit, other projects such as Synereo [27] and Akasha[8] are exploring the convergence of social media and decentralized content curation. Beyond blockchain-based social media platforms, coin-holder voting systems are present in decentralized platforms such as DAOs [7] and in different blockchain protocols [11, 20]. However, most of these systems use coin-holder voting processes to agree on a value or take a consensual decision.

3 Model

We first introduce some useful notation:

- We denote an ordered list of elements with $A = [e_1, \ldots, e_n]$ and the i-th element of the list with $A[i] = e_i$.
- Let $n \in \mathbb{N}^*$. $[n]$ denotes $\{1, 2, \ldots, n\}$.

3.1 Post list

▶ **Definition 1** (Post). *Let $N \in \mathbb{N}^*$. A post is defined as $P = (m, l)$, with $m \in [N], l \in [0,1]^N$.*

- ***Author.*** *The first element of a post is the id of its creator m.*
- ***Likability.*** *The likability of a post is defined as $l \in [0,1]^N$.*

N represents the number of voters (a.k.a. players). A post has a distinct likability in $[0, 1]$ for each player.

▶ **Definition 2** (Ideal Score of a post). *Let post $P = (m, l)$. We define the* ideal score *of P as* $\text{idealSc}(P) = \sum_{i=1}^{|l|} l_i$.

The ideal score of a post is a single number that represents its overall worth to the community. By using simple summation, we assume that the opinions of all players have the same weight.

▶ **Definition 3** (Post List). *Let $M \in \mathbb{N}^*$. A post list $\mathcal{P} = [P_1, \ldots, P_M]$ is an ordered list containing posts. It may be the case that two posts are identical.*

In the case of many UGC platforms, e.g. Steemit, there exists a feed (commonly named "Trending") that displays the same ordered posts for all users. In such an ordered list, posts placed closer to the top are more visible, since users typically consume content from top to bottom. We can thus measure the quality of an ordered list of posts by comparing it with a list that contains the same posts in decreasing order of ideal score.

▶ **Definition 4** (t-Ideal Post Order). *Let $\mathcal{P}$ a list of posts, $t \in [M]$. The property* $\textsc{Ideal}^t(\mathcal{P})$ *holds if*

$$\forall i < j \in [t], \text{idealSc}(\mathcal{P}[i]) \geq \text{idealSc}(\mathcal{P}[j]) \quad .$$

We say that $\mathcal{P}$ has a t-ideal rank *if* $\textsc{Ideal}^t(\mathcal{P})$ *holds and t is the maximum integer less or equal to M with this property.*

[8] `https://akasha.world/` Accessed: 2019-01-02

3.2 Post Voting System

We now define an abstract post-voting system. Such a system is defined through two Interactive Turing Machines (ITMs), $\mathcal{G}_{\text{Feed}}$ and Π_{honest}. The first controls the list of posts and aggregates votes, whereas one copy of the second ITM is instantiated for each player. $\mathcal{G}_{\text{Feed}}$ sends the post list to one player at a time, receives her vote and reorders the post list accordingly. The process is possibly repeated for many rounds.

A measure of the quality of a post-voting system is the t-ideal rank of the post list at the end of the process.

In a more general setting, some of the honest protocol instantiations may be replaced with an arbitrary ITM. A robust post-voting system should still produce a post list of high quality.

▶ **Definition 5** (Post-Voting System)**.** *Consider four PPT algorithms* Init*,* Aux*,* HandleVote *and* Vote*. The tuple $\mathcal{S}$ consisting of the four algorithms is a Post-Voting System. $\mathcal{S}$ parametrizes the following two ITMs:*

$\mathcal{G}_{\text{Feed}}$ is a global functionality that accepts two messages: `read`*, which responds with the current list of posts and* `vote`*, which can take various arguments and does whatever is defined in* HandleVote*.*

Π_{honest} is a protocol that sends `read` *and* `vote` *messages to $\mathcal{G}_{\text{Feed}}$ whenever it receives (*`activate`*) from $\mathcal{E}$.*

Algorithm 1 $\mathcal{G}_{\text{Feed}}$ (Init, Aux, HandleVote) $(\mathcal{P}, \text{initArgs})$.

1: Initialization:
2: $\quad\mathcal{U} \leftarrow \emptyset$ ▷ Set of players
3: $\quad$Init (initArgs)
4:
5: Upon receiving (`read`) from u_{pid}:
6: $\quad\mathcal{U} \leftarrow \mathcal{U} \cup \{u_{\text{pid}}\}$
7: $\quad$aux $\leftarrow$ Aux (u_{pid})
8: $\quad$Send (`posts`, $\mathcal{P}$, aux) to u_{pid}
9:
10: Upon receiving (`vote`, ballot) from u_{pid}:
11: $\quad$HandleVote(ballot)

Algorithm 2 Π_{honest} (Vote).

1: Upon receiving (`activate`) from $\mathcal{E}$:
2: $\quad$Send (`read`) to $\mathcal{G}_{\text{Feed}}$
3: $\quad$Wait for response (`posts`, $\mathcal{P}$, aux)
4: $\quad$ballot $\leftarrow$ Vote $(\mathcal{P}, \text{aux})$
5: $\quad$Send (`vote`, ballot) to $\mathcal{G}_{\text{Feed}}$

Players are activated by an Environment ITM that sends activation messages (Algorithm 2, line 1).

▶ **Definition 6** (Post-Voting System Activation Message)**.** *We define* $\texttt{act}_{\text{pid}}$ *as the message* $(\texttt{activate}, \text{pid})$*, sent to* u_{pid}*.*

▶ **Definition 7** (Execution Pattern). *Let* $N, R \in \mathbb{N}^*, N \geq 2$.

$$\text{ExecPat}_{N,R} = \left\{ \left(\texttt{act}_{\text{pid}_1}, \ldots, \texttt{act}_{\text{pid}_{NR}} \right) : \forall i \in [R], \forall k \in [N], \exists j \in [N] : \text{pid}_{(i-1)N+j} = k \right\} ,$$

i.e. activation messages are grouped in R *rounds and within each round each player is* `activated` *exactly once. The order of activations is not fixed.*

Let Environment $\mathcal{E}$ *that sends messages* $\text{msgs} = \left(\texttt{act}_{\text{pid}_1}, \ldots, \texttt{act}_{\text{pid}_n} \right)$ *sequentially. We say that* $\mathcal{E}$ respects $\text{ExecPat}_{N,R}$ *if* $\text{msgs} \in \text{ExecPat}_{N,R}$. *(Note: this implies that* $n = NR$*.)*

▶ **Definition 8** ((N, R, M, t)-convergence under honesty). *We say that a post-voting system* $\mathcal{S} = (\text{Init}, \text{Aux}, \text{HandleVote}, \text{Vote})$ (N, R, M, t)-converges under honesty *(or* t-converges under honesty for N players, R rounds and M posts*) if, for every input* $\mathcal{P}$ *such that* $|\mathcal{P}| = M$*, for every* $\mathcal{E}$ *that respects* $\text{ExecPat}_{N,R}$ *and given that all protocols execute* Π_{honest}*, it holds that after* $\mathcal{E}$ *completes its execution pattern,* $\mathcal{G}_{\text{Feed}}$ *contains a post list* $\mathcal{P}'$ *such that* $\text{Ideal}^t(\mathcal{P}')$ *is true.*

Note that concrete post voting systems may or may not give information such as the total number of rounds R to the players. This is decided in algorithm Aux.

We now give a high-level description of a concrete post voting system, based on the Steemit platform. According to this mechanism, each player is assigned a number of coins known as "Steem Power" (SP) that remains constant throughout the execution and another number called "Voting Power" (VP) in $[0, 1]$, initialized to 1. a and b are system-wide constants that roughly specify how influential a single vote is. A vote is a pair containing a post and a weight $w \in [0, 1]$. Upon receiving a list of posts, the honest player chooses to vote her most liked post amongst the top attSpan posts of the list. The weight w is chosen to be equal to the likability of the post. The functionality increases the score of the post by $\text{SP}(a \cdot \text{VP} \cdot w + b)$ and subsequently decreases the player's Voting Power by the same amount (but keeping it within the aforementioned bounds). Voting Power is replenished with time, at a rate defined by the parameter regen. The purpose of Voting Power is to "rate limit" votes.

▶ **Definition 9** (Steemit system). *The Steemit system is the post voting system* $\mathcal{S}$ *with parameters* $a, b, \text{regen} \in [0, 1] : a + b < 1, \left\lceil \frac{a+b}{\text{regen}} \right\rceil > 1, \text{attSpan} \in \mathbb{N}^*, \boldsymbol{SP} \in \mathbb{R}_+^N$. *The four parametrizing procedures can be found in Appendix B.*

▶ Remark 10. The constraint $a + b < 1$ ensures that a single vote of full weight cast by a player with full Voting Power does not completely deplete her Voting Power. The constraint $\left\lceil \frac{a+b}{\text{regen}} \right\rceil > 1$ excludes the degenerate case in which the regeneration of a single round is enough to fully replenish the Voting Power in all cases; in this case the purpose of Voting Power would be defeated.

▶ Remark 11. The Steem blockchain protocol defines $a = 0.02, b = 0.0001$ and $\text{regen} = \frac{3}{5 \cdot 24 \cdot 60 \cdot 60} = 0.00000694\bar{4}$, thus $\left\lceil \frac{a+b}{\text{regen}} \right\rceil = 2895$. A post can be voted for 7 days from its creation and at most one vote can be cast every 3 seconds, thus $R = \frac{7 \cdot 24 \cdot 60 \cdot 60}{3} = 201600$. We do not know why these particular parameters were chosen, but we conjecture that a, b and regen ensure users can vote often enough without abusing the system, 7 days is the time needed for the quality of a post to be determined and 3 seconds is the time needed for transactions to settle in the Steem blockchain.

▶ Remark 12. Note (Algorithm 6, lines 24-40) that an honest player attempts to vote for as many posts as possible and spreads her votes with the maximum distance between them. The purpose of this is to efficiently utilize the available Voting Power to "make her voice heard". Also, efficiently using Voting Power on the Steemit website increases the voter's curation reward [18].

▶ **Theorem 13.**

1. *If* $\exists i \neq j \in [N] : \mathrm{SP}_i \neq \mathrm{SP}_j$ *(i.e. if not all players have the same Steem Power) then Steemit does not* $(N, R, M, 1)$*-converge.*
2. *If* $\forall i \neq j \in [N], \mathrm{SP}_i = \mathrm{SP}_j$ *(i.e. if all players have the same Steem Power) and*
   a. $R - 1 \geq (M-1)\left\lceil \frac{a+b}{\text{regen}} \right\rceil$ *then Steemit* (N, R, M, M)*-converges.*
   b. $R - 1 < (M-1)\left\lceil \frac{a+b}{regen} \right\rceil$ *then Steemit does not* $(N, R, M, 1)$*-converge.*

Proof Sketch. When **SP** is not constant, we build a post list where the most liked post is not preferred by rich players and thus is not placed at the top. For a constant **SP**, when $R-1 \geq (M-1)\left\lceil \frac{a+b}{\text{regen}} \right\rceil$, there are enough rounds to ensure full regeneration of every player's Voting Power between two votes and thus the resulting post list reflects the true preferences of the players. In the opposite case, we can always craft a post list that exploits the fact that some votes are cast with reduced Voting Power in order to trick the system into placing a wrong post in the top position. ◀

See Appendix A for proof.

▶ **Corollary 14.** *The Steemit system parametrised according to Remark 11, for any number of players* $N \geq 2$*, constant* ***SP*** *and* $M \leq 70$ *posts* (N, R, M, M)*-converges. If* $M > 70$ *or* ***SP*** *is not constant, then there exists a list of posts such that the system does not* $(N, R, M, 1)$*-converge.*

4 Simulation

The previous outcomes are here complemented with experiments that verify our findings. We have implemented a simulation framework that realizes the execution of Steemit's post-voting system as defined above.

In particular, we consider two separate scenarios: First, we simulate the case when all players follow the prescribed honest strategy of Steemit, investigating how the curation quality of the system varies with the number of voting rounds. We successfully reproduce the result of Theorem 13, which implies that the system converges perfectly when a sufficient number of voting rounds is permitted, but otherwise the resulting list of posts may have a 0-ideal rank, i.e. the top post may not have the best ideal score. Moreover, we compare our t-convergence metric with previously used metrics of convergence based on correlation demonstrating that they are very closely aligned.

The second case measures how resilient is the curation quality of Steemit against dishonest agents. Since a creator is financially rewarded when her content is upvoted, she has incentive to promote her own posts. A combination of in-band methods (apart from striving to produce posts of higher quality) can help her to that end. Voting for one's own posts, refraining from voting posts created by others and obtaining Sybil accounts that only vote for her posts are only an indicative subset. We thus examine the quality of the resulting list when certain users do not follow the honest protocol, but apply the aforementioned self-promoting methods. We observe that even a single selfish player has a detrimental effect to the t-ideal rank of the post voting system. Furthermore, we measure the number of positions on the list that the selfish post gains with respect to the number of selfish players.

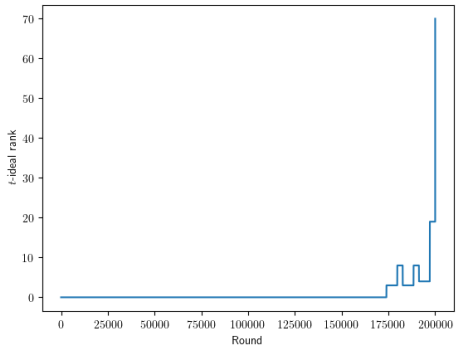
4.1 Methodology

We leverage three metrics to compare the curated list with the ideal list: Kendall's Tau [25], Spearman's Rho [37], and t-ideal rank.

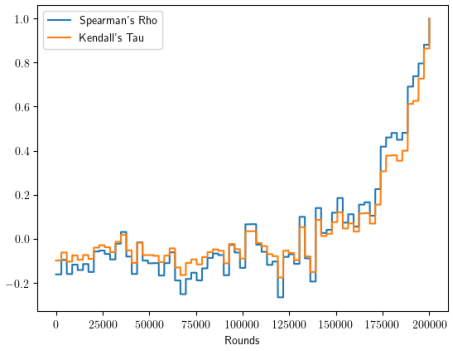
In addition to the t-ideal rank and the rank correlation coefficients used in the first scenario, in the case of dishonest participants we include a metric that measures the gains of the selfish players. In particular, the metric is defined as the difference between the real position of the "selfish" post after the execution and its ranking according to the ideal order. We are thus able to measure how advantageous is for users to behave selfishly. Furthermore, t-ideal rank informs us how this behavior affects the overall quality of curation of the platform.

4.2 Execution

In all simulations, the likabilities of all "honest" posts have been drawn from the $[0, 1]$-uniform distribution and all players have Steem Power equal to 1; we leave the case of variable Steem Power as future work.

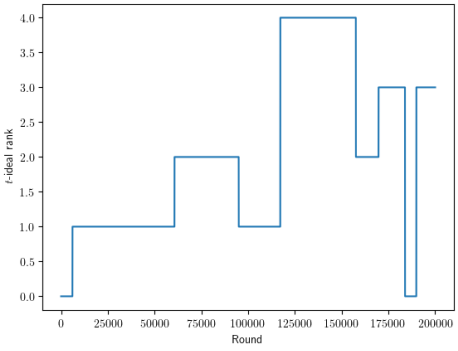


(a) t-ideal rank evolution.

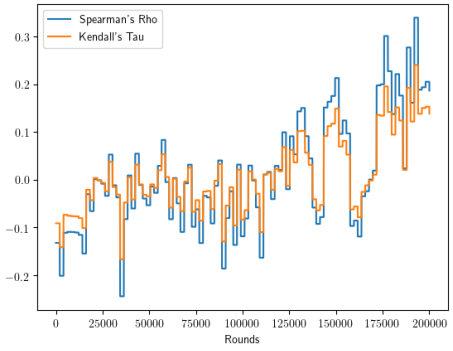


(b) Kendall's Tau and Spearman's Rho evolution.

Figure 1 270 honest players, 70 posts and 200.000 rounds.

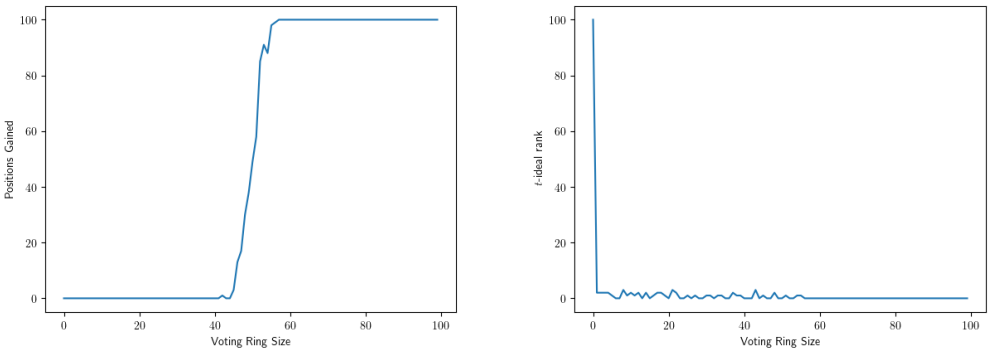


(a) t-ideal rank evolution.



(b) Kendall's Tau and Spearman's Rho evolution.

Figure 2 300 honest players, 100 posts and 200.000 rounds.



(a) Positions gained by selfish post. **(b)** t-ideal rank.

Figure 3 100 honest players, 100 posts and 0 to 100 selfish players.

4.2.1 Scenario A

As already mentioned, the results closely follow Theorem 13. Figures 1a and 1b show the t-ideal rank and Kendall's Tau coefficient respectively when the number of rounds is enough for all votes to be cast with full Voting Power. In particular, the parameters used are $a = \frac{1}{50}, b = 10^{-4}, \text{regen} = \frac{3}{5 \cdot 24 \cdot 60 \cdot 60}, R = 200000, \text{attSpan} = 10, N = 270$ and $M = 70$. (Observe that $R - 1 > (M - 1) \left\lceil \frac{a+b}{\text{regen}} \right\rceil$.)

As we can see, all three measures show that the real list converges rapidly to the ideal order at the very end of the execution; meanwhile, the quality of the list improves very slowly.

Figures 2a and 2b depict what happens when the rounds are not sufficient for all votes to be cast with full Voting Power. In particular, the corresponding simulation was executed with the same parameters, except for $M = 100$ and $N = 300$. (Observe that $R - 1 < (M - 1) \left\lceil \frac{a+b}{\text{regen}} \right\rceil$.)

Here we see that at the end of the execution, only the first three posts are correctly ordered. Regarding the rest of the list, both Kendall's Tau and Spearman's Rho coefficients show that the order of the posts improves only slightly throughout the execution of the simulation, ending up in a state of bad quality.

4.2.2 Scenario B: Selfish users

In order to understand how the presence of voting rings/Sybil accounts affects the curation quality, we simulate the execution of the game for various ring sizes, where ring members vote only for a particular "selfish" post. We fix the rest of the system parameters to handicap the selfish post. In particular, the voting rounds are sufficient for all votes to be cast with full Voting Power, the likability of the selfish post is 0 for all players and it is initially placed at the bottom of the post list. Define the gain of the post of the selfish players as its ideal position minus its final position. Figure 3a shows the gain of the selfish post for a varying number of selfish players, from 1 to 100. Figure 3b depicts the t-ideal rank of the resulting list at the same executions. The system parameters are $N = 101..200, a = \frac{1}{50}, b = 10^{-4}, \text{regen} = \frac{3}{5 \cdot 24 \cdot 60}, \text{attSpan} = 10, R = 5000$.

As we can see in Figure 3a, there is a cutoff point around which the selfish players quickly move from gaining no positions to overtaking all honest posts. The number of selfish players needed for this advantage is approximately half of the amount of honest ones. On the other hand, figure 3b shows that even a single selfish player can almost completely ruin the t-ideal rank of the result by only allowing a very small number of the best posts to be placed in the correct order.

5 Summary and Future Work

We have defined an abstract post-voting system, along with a particularization inspired by the Steemit platform. We proved the exact conditions on the Steemit system parameters under which it successfully curates arbitrary lists of posts. We provided the results of simulations of the execution of the voting procedure under various conditions. Both cases with only honest and mixed honest and selfish players were simulated. We conclude that the Voting Power mechanism of Steem and the fact that self-voting is a profitable strategy may hurt curation quality.

We have studied the curation properties of decentralized content curation platforms such as Steemit, obtaining new insights on the resilience of these systems. Some assumptions have been made in the presented model. Various relaxations of these assumptions constitute fertile ground for future work. First of all, the selfish strategy can be extended and refined in various ways. For example, voting rings can be allowed to create more than one posts in order to increase their rewards. Optimizing the number of posts and the vote allocation in this case would contribute towards a robust attack against the Steemit platform.

Selfish behavior is considered only in the simulation. Our analysis can be augmented with a review of games with selfish players and voting rings.

The addition of the economic factor invites the definition of utility functions and strategic behavior for the players. Its inclusion would imply the need for an expansion of our theorems and definitions to the strategic case, along with a full game-theoretic analysis. Furthermore, several possible refinements could be introduced; for example, the process of creating Sybil accounts could be associated with a monetary cost.

Last but not least, in our model, posts are created only at the beginning of the execution. A dynamic model in which posts can be created at any time and the execution continues indefinitely (as is the case in a real-world UGC system) is also interesting as a future direction.

References

1 Zeinab Abbassi, Nidhi Hegde, and Laurent Massoulié. Distributed content curation on the Web. *ACM Transactions on Internet Technology (TOIT)*, 14(2-3):9, 2014.

2 Ashton Anderson, Daniel Huttenlocher, Jon Kleinberg, and Jure Leskovec. Steering user behavior with badges. In *Proceedings of the 22nd international conference on World Wide Web*, pages 95–106. ACM, 2013.

3 Georgios Askalidis and Greg Stoddard. A theoretical analysis of crowdsourced content curation. In *The 3rd Workshop on Social Computing and User Generated Content*, volume 16, 2013.

4 Kelly Bergstrom. "Don't feed the troll": Shutting down debate about community expectations on Reddit. com. *First Monday*, 16(8), 2011.

5 Vitalik Buterin. Notes on Blockchain Governance. Accessed: 2019-01-02, 2017. URL: `https://vitalik.ca/general/2017/12/17/voting.html`.

6 Vincent Conitzer and Tuomas Sandholm. Communication complexity of common voting rules. In *Proceedings of the 6th ACM conference on Electronic commerce*, pages 78–87. ACM, 2005.

7 Philip Daian, Tyler Kell, Ian Miers, and Ari Juels. On-Chain Vote Buying and the Rise of Dark DAOs. Accessed: 2019-01-02, 2018. URL: `http://hackingdistributed.com/2018/07/02/on-chain-vote-buying/`.

8 dantheman. DPOS Consensus Algorithm - The Missing White Paper. URL: `https://steemit.com/dpos/@dantheman/dpos-consensus-algorithm-this-missing-white-paper` Accessed: 2019-04-02, 2017.

9 Anish Das Sarma, Atish Das Sarma, Sreenivas Gollapudi, and Rina Panigrahy. Ranking mechanisms in twitter-like forums. In *Proceedings of the third ACM international conference on Web search and data mining*, pages 21–30. ACM, 2010.

10 J. R. Douceur. The Sybil Attack. *International workshop on Peer-To-Peer Systems*, 2002.

11 Evan Duffield and Daniel Diaz. Dash: A PrivacyCentric CryptoCurrency. *Self-published*, 2015.

12 Fred Ehrsam. Blockchain Governance: Programming Our Future. *https://www.medium.com*, 2017. Accessed: 2019-01-02. URL: https://www.medium.com/@FEhrsam/blockchain-governance-programming-our-future-c3bfe30f2d74.

13 Andrea Forte and Amy Bruckman. Scaling consensus: Increasing decentralization in Wikipedia governance. In *Hawaii International Conference on System Sciences, Proceedings of the 41st Annual*, pages 157–157. IEEE, 2008.

14 Arpita Ghosh and Preston McAfee. Incentivizing high-quality user-generated content. In *Proceedings of the 20th international conference on World wide web*, pages 137–146. ACM, 2011.

15 Allan Gibbard. Manipulation of voting schemes: a general result. *Econometrica: journal of the Econometric Society*, pages 587–601, 1973.

16 Mike Goldin. Token-Curated Registries 1.0. Accessed: 2019-01-02, 2017. URL: https://medium.com/@ilovebagels/token-curated-registries-1-0-61a232f8dac7.

17 Oded Goldreich. The foundations of modern cryptography. In *Modern Cryptography, Probabilistic Proofs and Pseudorandomness*, pages 1–37. Springer, 1999.

18 Julián González. Author and Curator rewards in HF19. Accessed: 2019-01-02, 2018. URL: https://steemit.com/steemit/@jga/author-and-curator-rewards-in-hf19.

19 Julián González. Self-voters can achieve an interest of 248% APR!! URL: https://steemit.com/utopian-io/@jga/self-voters-can-achieve-an-interest-of-248-apr. Accessed: 2019-01-02, 2018.

20 LM Goodman. Tezos–a self-amending crypto-ledger White paper. URL: https://www.tezos.com/static/papers/white_paper.pdf, 2014.

21 Mangesh Gupte, MohammadTaghi Hajiaghayi, Lu Han, Liviu Iftode, Pravin Shankar, and Raluca M Ursu. News posting by strategic users in a social network. In *International Workshop on Internet and Network Economics*, pages 632–639. Springer, 2009.

22 Claude Hillinger. The case for utilitarian voting. *Homo Oeconomicus*, 22(3), 2005. . Accessed: 2019-04-01. URL: https://ssrn.com/abstract=878008.

23 Meir Kalech, Sarit Kraus, Gal A Kaminka, and Claudia V Goldman. Practical voting rules with partial information. *Autonomous Agents and Multi-Agent Systems*, 22(1):151–182, 2011.

24 Andreas M Kaplan and Michael Haenlein. Users of the world, unite! The challenges and opportunities of Social Media. *Business horizons*, 53(1):59–68, 2010.

25 Maurice G Kendall. *Rank Correlation Methods*, volume 9, page 68. Hafner Publishing Co., 2 edition, 1955. . Accessed: 2019-04-01. doi:10.1111/j.2044-8317.1956.tb00172.x.

26 Aggelos Kiayias, Benjamin Livshits, Andrés Monteoliva Mosteiro, and Orfeas Stefanos Thyfronitis Litos. A Puff of Steem: Security Analysis of Decentralized Content Curation. *arxiv.org*, 2018.

27 Dor Konforty, Yuval Adam, Daniel Estrada, and Lucius Gregory Meredith. Synereo: The Decentralized and Distributed Social Network. *Self-published*, 2015. Accessed: 2019-01-02. URL: https://pdfs.semanticscholar.org/253c/c4744e6b2b87f88e46188fe527982b19542e.pdf.

28 Jure Leskovec, Daniel P Huttenlocher, and Jon M Kleinberg. Governance in social media: A case study of the wikipedia promotion process. In *ICWSM*, pages 98–105, 2010.

29 Brian Neil Levine, Clay Shields, and N Boris Margolin. A survey of solutions to the sybil attack. *University of Massachusetts Amherst, Amherst, MA*, 7:224, 2006.

30 Yehuda Lindell and Jonathan Katz. *Introduction to modern cryptography*. Chapman and Hall/CRC, 2014.

31 Tyler Lu and Craig Boutilier. Robust approximation and incremental elicitation in voting protocols. In *IJCAI*, volume 1, pages 287–293, 2011.

32 Avner May, Augustin Chaintreau, Nitish Korula, and Silvio Lattanzi. Filter & follow: How social media foster content curation. In *ACM SIGMETRICS Performance Evaluation Review*, volume 42, pages 42–55. ACM, 2014.

33 Pasquale De Meo, Katarzyna Musial-Gabrys, Domenico Rosaci, Giuseppe ML Sarne, and Lora Aroyo. Using centrality measures to predict helpfulness-based reputation in trust networks. *ACM Transactions on Internet Technology (TOIT)*, 17(1):8, 2017.

34 Arash Molavi Kakhki, Chloe Kliman-Silver, and Alan Mislove. Iolaus: Securing online content rating systems. In *Proceedings of the 22nd international conference on World Wide Web*, pages 919–930. ACM, 2013.

35 Emilee Rader and Rebecca Gray. Understanding user beliefs about algorithmic curation in the Facebook news feed. In *Proceedings of the 33rd annual ACM conference on human factors in computing systems*, pages 173–182. ACM, 2015.

36 Fabian Schuh. Graphene Documentation. Accessed: 2019-04-02, 2018. URL: `https://media.readthedocs.org/pdf/docsbitsharesorg/master/docsbitsharesorg.pdf`.

37 Charles Spearman. The proof and measurement of association between two things. *The American journal of psychology*, 15(1):72–101, 1904.

38 Katarina Stanoevska-Slabeva, Vittoria Sacco, and Marco Giardina. Content Curation: a new form of gatewatching for social media. In *Proceedings of the 12th international symposium on online journalism*, 2012.

39 Mike Thelwall. Can social news websites pay for content and curation? The SteemIt cryptocurrency model. *Journal of Information Science*, page 0165551517748290, 2017.

40 Dinh Nguyen Tran, Bonan Min, Jinyang Li, and Lakshminarayanan Subramanian. Sybil-Resilient Online Content Voting. In *NSDI*, volume 9, pages 15–28, 2009.

41 Unknown. Steem: An incentivized, blockchain-based, public content platform. Accessed: 2019-04-02, 2017. URL: `https://steem.com/SteemWhitePaper.pdf`.

42 Unknown. Steem Whitepaper. Accessed: 2019-04-02, 2018. URL: `https://steem.io/steem-whitepaper.pdf`.

43 Bimal Viswanath, Mainack Mondal, Krishna P Gummadi, Alan Mislove, and Ansley Post. Canal: Scaling social network-based Sybil tolerance schemes. In *Proceedings of the 7th ACM european conference on Computer Systems*, pages 309–322. ACM, 2012.

44 Lirong Xia and Vincent Conitzer. Compilation Complexity of Common Voting Rules. In *AAAI*, 2010.

45 Haifeng Yu, Chenwei Shi, Michael Kaminsky, Phillip B Gibbons, and Feng Xiao. Dsybil: Optimal sybil-resistance for recommendation systems. In *2009 30th IEEE Symposium on Security and Privacy*, pages 283–298. IEEE, 2009.

46 Yingwu Zhu. Measurement and analysis of an online content voting network: a case study of Digg. In *Proceedings of the 19th international conference on World wide web*, pages 1039–1048. ACM, 2010.

A Proof of Theorem 13: Steem Convergence

Proof.

- Statement 1: Reorder the players such that $\mathrm{SP}_1 \geq \mathrm{SP}_2 \geq \cdots \geq \mathrm{SP}_N$. Let $k = \min_{j \in [N-1]} \{\mathrm{SP}_j \neq \mathrm{SP}_{j+1}\}$. We first cover the case when attSpan ≥ 2.
  Let[9]

$$\text{weakPost} = (\underbrace{0, \ldots, 0}_{k-1}, 1, \underbrace{0, \ldots, 0}_{N-k})$$

[9] We thank Heng Guo from the University of Edinburgh for this counterexample.

$$\text{strongPost} = (\underbrace{0,\ldots,0}_{k-1}, \frac{\text{SP}_k - \text{SP}_{k+1}}{2\text{SP}_k}, 1, \underbrace{0,\ldots,0}_{N-k-1})$$
$$\text{nullPost} = (\underbrace{0,\ldots,0}_{N})\mathcal{P} = [\text{weakPost}, \text{strongPost}, \underbrace{\text{nullPost},\ldots,\text{nullPost}}_{M-2}] \ .$$

We first note that $\text{SP}_k > \text{SP}_{k+1} \geq 0 \Rightarrow 0 \leq \frac{\text{SP}_k - \text{SP}_{k+1}}{2\text{SP}_k} \leq 1$, thus strongPost is a valid post. We then observe that

$$\forall i \in \{3,\ldots,M\}, \text{idealSc}\left(\mathcal{P}\left[i\right]\right) = 0 <$$
$$< \text{idealSc}\left(\mathcal{P}\left[1\right]\right) = 1 < 1 + \frac{\text{SP}_k - \text{SP}_{k+1}}{2\text{SP}_k} = \text{idealSc}\left(\mathcal{P}\left[2\right]\right) \ ,$$

thus $\forall \mathcal{P}'$ that contain the same posts as $\mathcal{P}$ and $\textsc{Ideal}^1\left(\mathcal{P}'\right)$ holds, it is $\mathcal{P}'\left[1\right] = \mathcal{P}\left[2\right]$. Since attSpan ≥ 2, all players apart from u_{k+1} vote for $\mathcal{P}\left[1\right]$ in the first round and for $\mathcal{P}\left[2\right]$ in the second, whereas u_{k+1} votes for $\mathcal{P}\left[2\right]$ in the first round and for $\mathcal{P}\left[1\right]$ in the second. Thus the two first posts will have been voted by all players by the end of the second round and their score will not change until the execution completes. We have:

$$\text{sc}_2\left(\mathcal{P}\left[1\right]\right) = \text{sc}_R\left(\mathcal{P}\left[1\right]\right) =$$
$$\sum_{j=1}^{k-1} \text{SP}_j b + \text{SP}_k\left(a+b\right) + \text{SP}_{k+1} \min\left\{b, \mathbf{VP}\text{reg}_{k+1,r_2}\right\} + \sum_{j=k+2}^{M} \text{SP}_j b \text{ and}$$
$$\text{sc}_2\left(\mathcal{P}\left[2\right]\right) = \text{sc}_R\left(\mathcal{P}\left[2\right]\right) =$$
$$\sum_{j=1}^{k-1} \text{SP}_j \min\left\{b, \mathbf{VP}\text{reg}_{j,r_2}\right\} +$$
$$\text{SP}_k \min\left\{a\frac{\text{SP}_k - \text{SP}_{k+1}}{2\text{SP}_k}\mathbf{VP}\text{reg}_{k,r_2} + b, \mathbf{VP}\text{reg}_{k,r_2}\right\} + \text{SP}_{k+1}\left(a+b\right) +$$
$$\sum_{j=k+2}^{M} \text{SP}_j \min\left\{b, \mathbf{VP}\text{reg}_{j,r_2}\right\} \Rightarrow$$

$$\text{sc}_R(\mathcal{P}\left[2\right]) \leq$$
$$\sum_{j=1}^{k-1} \text{SP}_j b + \text{SP}_k(a\frac{\text{SP}_k - \text{SP}_{k+1}}{2\text{SP}_k} + b) + \text{SP}_{k+1}\left(a+b\right) + \sum_{j=k+2}^{M} \text{SP}_j b \ .$$

In the case that $\mathbf{VP}\text{reg}_{k+1,r_2} \geq b$, it is

$$\text{sc}_R\left(\mathcal{P}\left[1\right]\right) = \sum_{j=1}^{k-1} \text{SP}_j b + \text{SP}_k\left(a+b\right) + \text{SP}_{k+1} b + \sum_{j=k+2}^{M} \text{SP}_j b >$$
$$\sum_{j=1}^{k-1} \text{SP}_j b + \text{SP}_k(a\frac{\text{SP}_k - \text{SP}_{k+1}}{2\text{SP}_k} + b) + \text{SP}_{k+1}\left(a+b\right) + \sum_{j=k+2}^{M} \text{SP}_j b \geq$$
$$\text{sc}_R\left(\mathcal{P}\left[2\right]\right) \Rightarrow \text{sc}_R\left(\mathcal{P}\left[1\right]\right) > \text{sc}_R\left(\mathcal{P}\left[2\right]\right) \ ,$$

thus $\textsc{Ideal}^1\left(\mathcal{P}'\right)$ does not hold.

Since u_{k+1} does not vote in any round between r_1 and r_2, and $r_2 \geq 2$, it is $\mathbf{VP}\text{reg}_{k+1,r_2} \geq 1 - a - b + \text{regen}$. Thus the case when $\mathbf{VP}\text{reg}_{k+1,r_2} < b$ can happen only when $b > 1 - a - b + \text{regen} \Leftrightarrow b > \frac{1-a+\text{regen}}{2}$. We now provide a counterexample for the case when $b > \frac{1-a+\text{regen}}{2}$.

Once more we order the players in descending Steem Power, like in the previous case. Once again $k = \min_{j \in [N-1]} \{\text{SP}_j \neq \text{SP}_{j+1}\}$ and we only care for the case when attSpan ≥ 2. Let $0 < \gamma < 1$ and

$$\text{weakPost} = (\underbrace{0, \ldots, 0}_{k-1}, 1, \frac{\gamma}{2}, \underbrace{0, \ldots, 0}_{N-k-1})$$

$$\text{strongPost} = (\underbrace{0, \ldots, 0}_{k-1}, \gamma, 1, \underbrace{0, \ldots, 0}_{N-k-1})$$

$$\text{nullPost} = (\underbrace{0, \ldots, 0}_{N})$$

$$\mathcal{P} = [\text{weakPost}, \text{strongPost}, \underbrace{\text{nullPost}, \ldots, \text{nullPost}}_{M-2}] \ .$$

We observe that $\forall i \in \{3, \ldots, M\}, \text{idealSc}(\mathcal{P}[i]) = 0 < \text{idealSc}(\mathcal{P}[1]) = 1 + \frac{\gamma}{2} < 1 + \gamma = \text{idealSc}(\mathcal{P}[2])$, thus $\forall \mathcal{P}'$ that contain the same posts as $\mathcal{P}$ and $\textsc{Ideal}^1(\mathcal{P}')$ holds, it is $\mathcal{P}'[1] = \mathcal{P}[2]$.

Since attSpan ≥ 2, all players apart from u_{k+1} vote for $\mathcal{P}[1]$ in the first round and for $\mathcal{P}[2]$ in the second, whereas u_{k+1} votes for $\mathcal{P}[2]$ in the first round and for $\mathcal{P}[1]$ in the second. Thus the two first posts will have been voted by all players by the end of the second round and their score will not change until the execution completes. We have:

$$\text{sc}_2(\mathcal{P}[1]) = \text{sc}_R(\mathcal{P}[1]) =$$

$$\sum_{j=1}^{k-1} \text{SP}_j b + \text{SP}_k(a+b) + \text{SP}_{k+1}\mathbf{VP}\text{reg}_{k+1,r_2} + \sum_{j=k+2}^{M} \text{SP}_j b \text{ and}$$

$$\text{sc}_2(\mathcal{P}[2]) = \text{sc}_R(\mathcal{P}[2]) =$$

$$\sum_{j=1}^{k-1} \text{SP}_j \min\{b, \mathbf{VP}\text{reg}_{j,r_2}\} + \text{SP}_k\mathbf{VP}\text{reg}_{k,r_2} + \text{SP}_{k+1}(a+b) +$$

$$\sum_{j=k+2}^{M} \text{SP}_j \min\{b, \mathbf{VP}\text{reg}_{j,r_2}\} \leq$$

$$\sum_{j=1}^{k-1} \text{SP}_j b + \text{SP}_k\mathbf{VP}\text{reg}_{k,r_2} + \text{SP}_{k+1}(a+b) + \sum_{j=k+2}^{M} \text{SP}_j b \ .$$

We note that $\mathbf{VP}\text{reg}_{k,r_2} = \mathbf{VP}\text{reg}_{k+1,r_2}$ because both u_k and u_{k+1} vote with full Voting Power in the first round. Let $\text{VP} = \mathbf{VP}\text{reg}_{k,r_2}$. We have

$$\text{SP}_k(a+b) + \text{SP}_{k+1}\text{VP} > \text{SP}_k\text{VP} + \text{SP}_{k+1}(a+b) \Leftrightarrow$$
$$\text{SP}_k(a+b) + \text{SP}_{k+1}\text{VP} - \text{SP}_k\text{VP} - \text{SP}_{k+1}(a+b) > 0 \Leftrightarrow$$
$$(a+b)(\text{SP}_k - \text{SP}_{k+1}) - \text{VP}(\text{SP}_k - \text{SP}_{k+1}) > 0 \Leftrightarrow$$
$$(\text{SP}_k - \text{SP}_{k+1})(a+b-\text{VP}) > 0$$

The last expression is true because $\text{SP}_k > \text{SP}_{k+1}$ and $\text{VP} < b$, thus the first expression is true as well. We can then deduce that $\text{sc}_R(\mathcal{P}[1]) > \text{sc}_R(\mathcal{P}[2])$, thus $\textsc{Ideal}^1(\mathcal{P}')$ does not hold. Please refer to the full version [26] for the case when attSpan $= 1$.

- Statement 2a: Suppose that

$$R-1 \geq (M-1)\left\lceil \frac{a+b}{\text{regen}} \right\rceil \ . \tag{1}$$

Observe that

$$(1) \Rightarrow \frac{R-1}{M-1} \geq \left\lceil \frac{a+b}{\text{regen}} \right\rceil \underset{\text{integer}}{\overset{\text{rhs}}{\Rightarrow}} \left\lfloor \frac{R-1}{M-1} \right\rfloor \geq \left\lceil \frac{a+b}{\text{regen}} \right\rceil \ . \tag{2}$$

Let $\text{pid} \in [N]$. From (1) we deduce that $R \geq M$ and according to VoteThisRound in Algorithm 6, u_{pid} votes non-null in rounds $(r_1, \ldots, r_M)$ with $r_i = \left\lfloor (i-1)\,\frac{R-1}{M-1} \right\rfloor + 1$. We define the following:

$$\begin{aligned}
&k \in \mathbb{N}, w \in \mathbb{R} \ , \\
&n \in \mathbb{Z}, p \in [0,1) : (k-1)\,w = n + p \ , \\
&m \in \mathbb{Z}, q \in [0,1) : w = m + q \ .
\end{aligned}$$

We have

$$\lfloor (k-1)\,w \rfloor = n \ , \tag{3}$$

$$\lfloor kw \rfloor = \begin{cases} n+m, & p+q<1 \\ n+m+1, & p+q \geq 1 \text{ (impossible if } p=0) \end{cases} \tag{4}$$

$$\lfloor w \rfloor = m \tag{5}$$

$$\lceil w \rceil = \begin{cases} m, & p=0 \\ m+1, & p>0 \end{cases} \tag{6}$$

$$\begin{aligned}
&(3),(4),(5),(6), p+q<2 \Rightarrow \\
&\lfloor kw \rfloor \in \{ \lfloor (k-1)\,w \rfloor + \lfloor w \rfloor, \lfloor (k-1)\,w \rfloor + \lceil w \rceil \}
\end{aligned} \tag{7}$$

From (7) we deduce that

$$\forall i \in [M] \setminus \{1\}, r_i \in \{ r_{i-1} + \left\lfloor \frac{R-1}{M-1} \right\rfloor, r_{i-1} + \left\lceil \frac{R-1}{M-1} \right\rceil \} \ . \tag{8}$$

From (2) and (8) we have that $\forall i \in [M-1], r_{i+1} - r_i \geq \left\lceil \frac{a+b}{\text{regen}} \right\rceil$. We will now prove by induction that $\forall i \in [M], \mathbf{VP}_{\text{pid},r_i} = 1$.

- For $i = 1, \mathbf{VP}_{\text{pid},1} = 1$ (Algorithm 3, line 4).
- Let $\mathbf{VP}_{\text{pid},r_i} = 1$. Until r_{i+1}, a single non-null vote is cast by u_{pid}, which reduces $\mathbf{VP}_{\text{pid}}$ by at most $a+b$ (Algorithm 5, line 7) and at least $\left\lceil \frac{a+b}{\text{regen}} \right\rceil$ regenerations, each of which replenishes $\mathbf{VP}_{\text{pid}}$ by regen. Thus

$$\mathbf{VP}_{\text{pid},r_{i+1}} \geq \min\{ \mathbf{VP}_{\text{pid},r_i} - a - b + \text{regen} \left\lceil \frac{a+b}{\text{regen}} \right\rceil, 1 \} \geq 1 \ .$$

But $\mathbf{VP}_{\text{pid}}$ cannot exceed 1 (line 4), thus $\mathbf{VP}_{\text{pid},r_{i+1}} = 1$.

Since the above holds for every pid $\in [N]$, it holds that at the end of the execution, all votes have been cast with full Voting Power, thus $\forall i \in [M], \mathrm{sc}_R(\mathcal{P}[i]) = Nb + a \sum_{\mathrm{pid}=1}^{N} \mathcal{P}[i]_{\mathrm{pid}}$ and the posts in $\mathcal{P}_R$ are sorted by decreasing score (Algorithm 5, line 20). We observe that

$$\forall i \neq j \in [M], \mathrm{idealSc}(\mathcal{P}[i]) > \mathrm{idealSc}(\mathcal{P}[j]) \Rightarrow$$
$$\sum_{\mathrm{pid}=1}^{N} \mathcal{P}[i]_{\mathrm{pid}} > \sum_{\mathrm{pid}=1}^{N} \mathcal{P}[j]_{\mathrm{pid}} \Rightarrow$$
$$Nb + a \sum_{\mathrm{pid}=1}^{N} \mathcal{P}[i]_{\mathrm{pid}} > Nb + a \sum_{\mathrm{pid}=1}^{N} \mathcal{P}[j]_{\mathrm{pid}} \quad .$$

Therefore all posts will be ordered according to their ideal scores; put otherwise, $\textsc{IdealScore}^M(\mathcal{P}_R)$ holds.

- Statement 2b: Suppose that

$$R - 1 < (M-1) \left\lceil \frac{a+b}{\mathrm{regen}} \right\rceil \quad . \tag{9}$$

Several lists of posts will be defined in the rest of the proof. Given that, when all players are honest, the creator of a post is irrelevant, we omit the creator from the definition of posts to facilitate the exposition. Thus every post will be defined as a tuple of likabilities. First, we consider the case when

$$\mathrm{attSpan} + R \leq M \quad . \tag{10}$$

In this case, no player can ever vote for the last post, as we will show now. First of all, (10) $\Rightarrow R < M$, thus all players cast R votes in total. Let pid $\in N, i \in [R]$ and $v_{\mathrm{pid},i}$ the index of the last post that has ever been in u_{pid}'s attention span until the end of round i, according to the ordering of $\mathcal{P}$. It is $v_{\mathrm{pid},1} = \mathrm{attSpan}$ and $\forall i \in [R] \backslash \{1\}, v_{\mathrm{pid},i} = v_{\mathrm{pid},i-1}+1$, since in every round u_{pid} votes for a single post and the first unvoted post of the list is added to their attention span. Note that, since this mechanism is the same for all players, the same unvoted post is added to all players' attention span at every round. Thus $\forall \mathrm{pid} \in N, v_{\mathrm{pid},R} = \mathrm{attSpan} + R - 1 \overset{(10)}{<} M$. We deduce that no player has ever the chance to vote for the last post. The above observation naturally leads us to the following counterexample: Let

$$\mathrm{strongPost} = (\underbrace{1, \ldots, 1}_{N}), \mathrm{nullPost} = (\underbrace{0, \ldots, 0}_{N})$$
$$\mathcal{P} = [\underbrace{\mathrm{nullPost}, \ldots, \mathrm{nullPost}}_{M-1}, \mathrm{strongPost}]$$

$\forall i \in [M-1]$, it is $\mathrm{idealSc}(\mathcal{P}[M]) > \mathrm{idealSc}(\mathcal{P}[i])$, thus $\forall \mathcal{P}'$ that contain the same posts as $\mathcal{P}$ and $\textsc{Ideal}^1(\mathcal{P}')$ holds, it is $\mathcal{P}'[1] = \mathcal{P}[M]$. However, since the last post is not voted by any player and the first post is voted by at least one player, it is $\mathrm{sc}_R(\mathcal{P}[1]) > \mathrm{sc}_R(\mathcal{P}[M])$, thus $\textsc{Ideal}^1(\mathcal{P}_R)$ does not hold.
We now move on to the case when $\mathrm{attSpan} + R > M$. Let $V = \min\{R, M\}$. Each player casts exactly V votes. Consider $\mathcal{P}^1 = 1^{M \times N}$ and pid $\in [N]$. Let

$$i \in [V] : \left(\mathbf{VP}\mathrm{reg}_{\mathrm{pid},r_i} < 1 \wedge \nexists i' < i : \mathbf{VP}\mathrm{reg}_{\mathrm{pid},r_{i'}} < 1 \right) \quad ,$$

i.e. i is the first round in which u_{pid} votes with less than full Voting Power. Such a round exists in every case as we will show now. Note that, since the first round is a voting round and the Voting Power of all players is full at the beginning, if i exists it is $i \geq 2$.

- If $R \geq M$, it is $V = M$.
  If $\nexists i \in [M] : \left(\mathbf{VP}\text{reg}_{\text{pid},r_i} < 1 \wedge \nexists i' < i : \mathbf{VP}\text{reg}_{\text{pid},r_{i'}} < 1\right)$, then we have that $\forall i \in [M], \mathbf{VP}\text{reg}_{\text{pid},r_i} = 1 \Rightarrow \forall i \in [M] \setminus \{1\}, r_i \geq r_{i-1} + \left\lceil \frac{a+b}{\text{regen}} \right\rceil$ to have enough rounds to replenish the Voting Power after a full-weight, full-Voting Power vote. Thus $r_M \geq 1 + (M-1)\left\lceil \frac{a+b}{\text{regen}} \right\rceil > R$, contradiction.
- If $R < M$, every player votes on all rounds, thus $r_2 = 2$. Note that

$$\left\lceil \frac{a+b}{\text{regen}} \right\rceil \geq 2 \Rightarrow \frac{a+b}{\text{regen}} > 1 \Rightarrow a + b > \text{regen} \ . \tag{11}$$

  Thus $\forall \text{pid} \in [N], \mathbf{VP}\text{reg}_{\text{pid},r_2} = 1 - a - b + \text{regen} \overset{(11)}{<} 1$, thus $i = 2$.

We proved that i exists. Since all players follow the same voting pattern, the Voting Power of all players in each round is the same. Let $\text{rVP} = \mathbf{VP}\text{reg}_{1,r_i}$. Assume that $\text{attSpan} < i \vee i > 2$. Please refer to the full version [26] for the case when $\text{attSpan} \geq i \wedge i = 2$. In case N is even, let $0 < \gamma < 0, 0 < \epsilon < \gamma\,(1 - \text{rVP})$,

$$\text{weakPost} = (\underbrace{1, \ldots, 1}_{N/2}, \underbrace{\gamma - \epsilon, \ldots, \gamma - \epsilon}_{N/2}) \ ,$$

$$\text{strongPost} = (\underbrace{\gamma, \ldots, \gamma}_{N/2}, \underbrace{1, \ldots, 1}_{N/2}), \text{nullPost} = (\underbrace{0, \ldots, 0}_{N}) \ ,$$

$$\mathcal{P} = [\underbrace{\text{weakPost}, \ldots, \text{weakPost}}_{i-1}, \text{strongPost}, \underbrace{\text{nullPost}, \ldots, \text{nullPost}}_{M-i}] \ .$$

First of all, it is

$$\forall j \in [i-1], \text{idealSc}\left(\mathcal{P}[j]\right) = \frac{N}{2}\left(1 + \gamma - \epsilon\right) <$$
$$< \frac{N}{2}\left(1 + \gamma\right) = \text{idealSc}\left(\mathcal{P}[i]\right)$$

and $\forall j \in \{i+1, \ldots, M\}, \text{idealSc}\left(\mathcal{P}[j]\right) = 0 < \text{idealSc}\left(\mathcal{P}[i]\right)$, thus the strong post has strictly the highest ideal score of all posts and as a result, $\forall \mathcal{P}'$ that contains the same posts as $\mathcal{P}$ and $\textsc{Ideal}^1\left(\mathcal{P}'\right)$ holds, it is $\mathcal{P}'[1] = \mathcal{P}[i]$.
We observe that all players like both weak and strong posts more than null posts, thus no player will vote for a null post unless her attention span contains only null posts. This can happen in two cases: First, if the player has not yet voted for all non-null posts, but the first attSpan posts of the list, excluding already voted posts, are null posts. Second, if the player has already voted for all non-null posts. For a null post to rank higher than a non-null one, it must be true that there exists one player that has cast the first vote for the null post. However, since the null posts are initially at the bottom of the list and it is impossible for a post to improve its ranking before it is voted, we deduce that this first vote can be cast only after the voter has voted for all non-null posts. We deduce that all players vote for all non-null posts before voting for any null post.
We will now see that the first $\frac{N}{2}$ players vote first for all weak posts and then for the strong post. These players like the weak posts more than the strong post. As we saw, they will not vote any null post before voting for all non-null ones. If $\text{attSpan} > 1$ they

vote for the strong post only when all other posts in their attention span are null ones and thus they will have voted for all weak posts already. If attSpan = 1 and since no post can increase its position before being voted, the strong post will become "visible" for all players only once they have voted for all weak posts. Thus in both cases the first $\frac{N}{2}$ players vote for the strong post only after they have voted for all weak posts first.

The two previous results combined prove that the first $\frac{N}{2}$ players vote for the strong post in round r_i exactly. We also observe that these players have experienced the exact same Voting Power reduction and regeneration as in the case of $\mathcal{P}^1$ since they voted only for posts with likability 1, thus in round r_i their Voting Power after regeneration is exactly the same as in the case of $\mathcal{P}^1 : \forall \text{pid} \in \left[\frac{N}{2}\right], \mathbf{VP}\text{reg}_{\text{pid},r_i} = \text{rVP}$.

We observe that the first $\frac{N}{2}$ players vote for all weak posts with full Voting Power. As for the last $\frac{N}{2}$ players, we observe that, if attSpan $< i$, they all vote for the first weak post of the list in the first round, and thus with full Voting Power. If attSpan $\geq i$ and $i > 2$, they vote for the strong post in the first round and for the first weak post in r_2 with full Voting Power. Thus in all cases the last $\frac{N}{2}$ players vote for the first weak post with full Voting Power. Therefore, the score of the first weak post at the end of the execution is $\text{sc}_R\left(\mathcal{P}\left[1\right]\right) = \frac{N}{2}\left(a+b\right) + \frac{N}{2}\left(\left(\gamma-\epsilon\right)a+b\right)$.

On the other hand, at the end of the execution the strong post has been voted by the first $\frac{N}{2}$ players with rVP Voting Power and by the last $\frac{N}{2}$ players with at most full Voting Power, thus its final score will be at most $\text{sc}_R\left(\mathcal{P}\left[i\right]\right) \leq \frac{N}{2}\left(\text{rVP}\cdot\gamma a+b\right) + \frac{N}{2}\left(a+b\right)$. It is

$$
\begin{aligned}
&\epsilon < \gamma\left(1-\text{rVP}\right) \Rightarrow \\
&\frac{N}{2}\left(\text{rVP}\cdot\gamma a+b\right) + \frac{N}{2}\left(a+b\right) < \frac{N}{2}\left(a+b\right) + \frac{N}{2}\left(\left(\gamma-\epsilon\right)a+b\right) \Rightarrow \\
&\text{sc}_R\left(\mathcal{P}\left[i\right]\right) < \text{sc}_R\left(\mathcal{P}\left[1\right]\right) \quad .
\end{aligned}
$$

Thus $\mathcal{P}_R\left[1\right] \neq \mathcal{P}\left[i\right]$ and $\text{Ideal}^1\left(\mathcal{P}_R\right)$ does not hold.

As for the case when N is odd, let $0 < \epsilon < \gamma\frac{N-3}{N-1}\left(1-\text{rVP}\right)$. In this case, we assume that the likability of the first i posts (weak and strong) for the additional player is γ, whereas the likability of the last $M-i$ posts (the null posts) is 0. This means that the additional player votes first for the weak and strong posts and then for the null posts. The rest of the likabilities remain as in the case when N is even. We observe that the ideal score of the strong post is still strictly higher than the rest. Furthermore, since the additional player votes for the first weak post within the first i voting rounds, her Voting Power at the time of this vote will be at least rVP. We thus have the following bounds for the scores:

$$
\begin{aligned}
&\text{sc}_R\left(\mathcal{P}\left[i\right]\right) \leq \frac{N-1}{2}\left(\text{rVP}\cdot\gamma a+b\right) + \frac{N-1}{2}\left(a+b\right) + \gamma a + b \ , \\
&\text{sc}_R\left(\mathcal{P}\left[1\right]\right) \geq \frac{N-1}{2}\left(a+b\right) + \frac{N-1}{2}\left(\left(\gamma-\epsilon\right)a+b\right) + \text{rVP}\cdot\gamma a + b \ .
\end{aligned}
$$

Given the bounds of ϵ, it is $\text{sc}_R\left(\mathcal{P}\left[i\right]\right) < \text{sc}_R\left(\mathcal{P}\left[1\right]\right)$, thus $\text{Ideal}^1\left(\mathcal{P}_R\right)$ does not hold. ◀

B Steem Post Voting System Procedures

Algorithm 3 INIT (attSpan, a, b, regen, R, **SP**).

Store input parameters as constants
$r \leftarrow 1$
lastVoted $\leftarrow (0, \ldots, 0) \in (\mathbb{N}^*)^N$
VP $\leftarrow (1, \ldots, 1) \in [0, 1]^N$
scores $\leftarrow (0, \ldots, 0) \in (\mathbb{R}^+)^M$

Algorithm 4 AUX.

return (attSpan, a, b, r, regen, R, **SP**)

Algorithm 5 HANDLEVOTE (ballot, u_{pid}).

if $\text{lastVoted}_{\text{pid}} \neq r$ **then** ▷ One vote per player per round
  $\mathbf{VP}_{\text{pid},r} \leftarrow \mathbf{VP}_{\text{pid}}$ ▷ For proofs
  $\mathbf{VP}_{\text{pid}} \leftarrow \max\{\mathbf{VP}_{\text{pid}} + \text{regen}, 1\}$
  $\mathbf{VPreg}_{\text{pid},r} \leftarrow \mathbf{VP}_{\text{pid}}$ ▷ For proofs
  if ballot $\neq$ **null then**
    Parse ballot as (P, weight)
    $\text{cost} \leftarrow a \cdot \mathbf{VP}_{\text{pid}} \cdot \text{weight} + b$
    if $\mathbf{VP}_{\text{pid}} - \text{cost} \geq 0$ **then**
      $\text{score} \leftarrow \text{cost} \cdot \mathbf{SP}_{\text{pid}}$
      $\mathbf{VP}_{\text{pid}} \leftarrow \mathbf{VP}_{\text{pid}} - \text{cost}$
    else
      $\text{score} \leftarrow \mathbf{VP}_{\text{pid}} \cdot \mathbf{SP}_{\text{pid}}$
      $\mathbf{VP}_{\text{pid}} \leftarrow 0$
    end if
    $\text{scores}_P \leftarrow \text{scores}_P + \text{score}$
  end if
  $\text{lastVoted}_{\text{pid}} \leftarrow r$
end if
if $\forall i \in [N], \text{lastVoted}_i = r$ **then** ▷ round over
  $\mathcal{P} \leftarrow$ ORDER $(\mathcal{P}, \text{scores})$ ▷ order posts by votes
  $\mathcal{P}_r \leftarrow \mathcal{P}$ ▷ For proofs
  $r \leftarrow r + 1$
end if

Algorithm 6 VOTE $(\mathcal{P}, \text{aux})$.

Store aux contents as constants
voteRounds $\leftarrow$ VOTEROUNDS $(R, |\mathcal{P}|)$
if VOTETHISROUND $(r, |\mathcal{P}|) = \text{yes}$ **then**
    top $\leftarrow$ CHOOSETOPPOSTS (attSpan, $\mathcal{P}$, votedPosts)
    $(i, l) \leftarrow \underset{(i,l)\in\text{top}}{\text{argmax}} \{l_{\text{pid}}\}[1]$
    votedPosts $\leftarrow$ votedPosts $\cup (i, l)$
    return $((i, l), l_{\text{pid}})$
else
    return null
end if

function CHOOSETOPPOSTS(attSpan, $\mathcal{P}$, votedPosts)
    res $\leftarrow \emptyset$
    idx $\leftarrow 1$
    while |res| $<$ attSpan & idx $\leq |\mathcal{P}|$ **do**
        if $\mathcal{P}$ [idx] $\notin$ votedPosts **then** ▷ One vote per post per player
            res $\leftarrow$ res $\cup \{\mathcal{P}\text{ [idx]}\}$
        end if
        idx $\leftarrow$ idx $+ 1$
    end while
    return res
end function

function VOTETHISROUND(r, M)
    if $R < M$ **then**
        return yes
    else if $r \in$ voteRounds **then**
        return yes
    else
        return no
    end if
end function

function VOTEROUNDS(R, M)
    voteRounds $\leftarrow \emptyset$
    for $i = 1$ to M **do**
        voteRounds $\leftarrow$ voteRounds $\cup \left\{1 + \left\lfloor (i-1)\, \frac{R-1}{M-1} \right\rfloor\right\}$
    end for
    return voteRounds
end function

An Empirical Study of Speculative Concurrency in Ethereum Smart Contracts

Vikram Saraph
Department of Computer Science, Brown University, USA
vsaraph@cs.brown.edu

Maurice Herlihy
Department of Computer Science, Brown University, USA
mph@cs.brown.edu

Abstract

We use historical data to estimate the potential benefit of speculative techniques for executing Ethereum smart contracts in parallel. We replay transaction traces of sampled blocks from the Ethereum blockchain over time, using a simple speculative execution engine. In this engine, miners attempt to execute all transactions in a block in parallel, rolling back those that cause data conflicts. Aborted transactions are then executed sequentially. Validators execute the same schedule as miners.

We find that our speculative technique yields estimated speed-ups starting at about 8-fold in 2016, declining to about 2-fold at the end of 2017, where speed-up is measured using either gas costs or instruction counts. We also observe that a small set of contracts are responsible for many data conflicts resulting from speculative concurrent execution.

2012 ACM Subject Classification Computing methodologies → Parallel computing methodologies

Keywords and phrases Blockchains, Smart Contracts

Digital Object Identifier 10.4230/OASIcs.Tokenomics.2019.4

Related Version `https://arxiv.org/abs/1901.01376`

1 Introduction

A *blockchain* is a distributed data structure that implements a *ledger*: a tamper-proof, widely-accessible, append-only sequence of *transactions*. Blockchains form the basis for cryptocurrencies [8, 10, 11, 12, 13] and other applications that must maintain a shared state in the absence of a trusted central authority. In the Ethereum [8] blockchain, for example, if Alice wants to send a coin to Bob, she broadcasts her transaction to one or more *miners*, who package transactions into *blocks*, and then undertake a consensus protocol to agree on which block should be appended next to the shared blockchain. A *validator* is any party who reads the blockchain state and checks it for correctness. Miners are validators, of course, but so is any party who needs to query the blockchain state.

In many blockchain systems, client transactions can invoke scripts, often called *smart contracts*, or just *contracts*, that perform logic needed to support complex services such as trading, voting, and managing tokens. Here, we focus on Ethereum-style smart contracts.

Ethereum's smart contracts present a concurrency challenge. To reconstruct the blockchain's current state, each validator must re-execute, in a sequential, one-at-a-time order, every call to every smart contract. Such sequential validation is unattractive because it fails to exploit the concurrency provided by modern multicore architectures. Simply executing those calls in parallel is unsafe, because there may be dependencies between contracts: if one contract depends on the results of another, then those contracts must be executed in the same order by every validator. Because the Ethereum smart contract language is Turing-complete, and because contracts can reference one another through untyped function pointers, static analysis is unlikely to be broadly effective.

The most promising approach to concurrent execution of smart contracts is *speculation* [1, 4, 7]: the virtual machine executes contract calls in parallel against the current state, tracking each transaction's *read set* and *write set* (memory locations read and written). Writes to memory are intercepted and buffered. Two transactions *conflict* if they access the same memory location, and one access is a write. For every pair of conflicting transactions, one is discarded, and the other is committed. Speculative techniques typically work well when conflicts are rare, but perform poorly when conflicts are common.

1.1 Contributions

How well does speculative concurrency work for smart contract execution? This paper makes the following contributions. We exploit publicly-available historical data to estimate conflict rates in an existing blockchain. This methodology, replaying the historical transaction record against proposed alternative run-times, could be a useful model for other blockchain-centered investigations. This study is *exploratory*: it aspires to provide a relatively fast and cheap estimate of how well certain strategies are likely to do in practice, with the goal of focusing future research attention in directions more likely to be productive. Of course, an exploratory study necessarily employs sampling, estimation, and approximation.

As described below in more detail, we re-execute blocks sampled from the Ethereum blockchain against a simple speculative execution engine. This engine has two phases: in the first (concurrent) phase, all transactions are run in parallel. In the second (sequential) phase, transactions observed to conflict are discarded and re-run in one-at-a-time order. This execution strategy produced speed-ups ranging from about 8-fold for blocks sampled from 2016, gradually declining to about 2-fold for blocks sampled from 2017.

This study makes the following observations.

- Even simple speculative strategies yield non-trivial speed-ups.
- Over time, however, these speed-ups declined as transaction traffic increased.
- Distinguishing between reads and writes is important: treating a transaction's read and write sets as a single conflict set substantially increases conflict rates.
- More aggressive speculative strategies, such as running multiple concurrent phases, yield little additional benefit.
- Accurate static conflict analysis may yield a modest benefit.
- Increasing the number of cores in the simulated virtual machine from 16 to 64 improved speed-ups, but there was little improvement above 64 cores.
- In high-contention periods, most contention resulted from a very small number of popular contracts.

These observations suggest some directions for further research.

- In periods of high contention, most conflict is caused by a small number of very popular contracts. Today, contract writers have no motivation for avoiding such conflicts. It could be productive to devise incentives, perhaps in the form of reduced gas prices, for contracts that produce fewer data conflicts.
- Many data conflicts, such as crediting and debiting account balances, are probably artifacts of defining conflict naïvely in terms of read-write sets. Perhaps conflicts could be reduced by extending the virtual machine to provide explicit support for common commutative operations such as credits and debits.

1.2 Methodology in Brief

We replay each transaction in a block, computing each transaction's read and write sets. We then greedily sort the transactions into two bins: the *concurrent bin* holds transactions that do not conflict with any other transaction already in the concurrent bin, and the *sequential bin* holds the rest.

We then estimate the elapsed time required to (1) execute the concurrent bin transactions in parallel (including the cost of detecting and discarding conflicting transactions), followed by (2) sequentially executing the sequential bin transactions.

Since we do not have a parallel EVM implementation to test, we estimate a transaction's running time in two ways: either by the *gas* it consumed, or by the number of Ethereum Virtual Machine (EVM) bytecode instructions executed. (Both measures are easy to compute, and yield similar results.) The speed-up is the ratio between the estimated elapsed times for the sequential executions versus the longest speculative execution.

In the next section we describe related work. In Section 3, we outline Ethereum's architecture, smart contracts, and all relevant terminology. Section 4 describes the setup of our empirical study along with statistics summarizing observed results, while in Section 5, we consider various alternatives to the baseline setup. Finally, in Sections 6 and 7, we discuss conclusions and potential future directions for extending this work.

2 Related Work

Smart contracts were first proposed by Szabo [15].

Bitcoin [12] includes a scripting language of limited power. Ethereum [8] is perhaps the most widely used smart contract platform, running on a quasi-Turing-complete virtual machine. Solidity [14] is the most popular programming language for the Ethereum virtual machine. Other blockchains that support smart contracts include Corda [5] and Cardano [9].

Hyperledger Fabric [4] is a permissioned blockchain where transactions (calls to smart contracts) are executed speculatively in parallel against the latest committed state. Transactions' read and write sets are recorded and compared, and conflicting contracts are discarded.

Dickerson et al. [7] have proposed a speculative execution model where miners dynamically construct a fork-join schedule that allows concurrent executions without violating transaction dependencies. Anjana *et al.* [1] propose a way to extend this approach to lock-free executions.

3 Ethereum Smart Contracts

In this section, we provide some background on Ethereum smart contracts, mining, and validation of Ethereum blocks.

3.1 The Architecture

In Ethereum, as in other blockchains, multiple *nodes* follow a common protocol in which *transactions* from *clients* are packaged into *blocks*, and nodes use a consensus protocol to agree on successive blocks. Each block includes a cryptographic hash of its predecessor, making it difficult to tamper with the ledger.

Each client has ownership of one or more *accounts*, so that each transaction occurs between a *sender* account and a *recipient* account. The majority of transactions are one of two kinds: either a *value transfer*, which is a purely monetary transfer of *ether* from sender to recipient, or a *contract call*, where the sender account makes a call to code associated with the recipient account.

```
1  contract StorageInterface {
2      mapping (uint => uint) storage;
3
4      function getValue(uint key) returns (uint) {
5          return storage[key];
6      }
7
8      function setValue(uint key, uint value) {
9          storage[key] = value;
10     }
11 }
```

(a) This code is a simple smart contract written in the Solidity language. The contract, StorageInterface, contains a Solidity mapping named storage, and has functions getValue and setValue. Mappings are essentially hash tables that store key-value pairs, and are the primary means of accessing Ethereum contract storage. Figure 1b shows a snippet of the bytecode after compiling the Solidity contract. The bytecode is what is published in the blockchain.

```
247  ...
248  JUMPDEST
249  DUP1
250  PUSH1 0x0
251  DUP1
252  DUP5
253  DUP2
254  MSTORE
255  PUSH1 0x20
256  ADD
257  SWAP1
258  DUP2
259  MSTORE
260  PUSH1 0x20
261  ADD
262  PUSH1 0x0
263  KECCAK256
264  DUP2
265  SWAP1
266  SSTORE
267  POP
268  ...
```

(b) Bytecode resulting from compiling the contract.

Figure 1 An example Solidity smart contract and a fragment of the corresponding bytecode.

Some Ethereum accounts, in addition to maintaining a balance of ether, possess associated code called a *smart contract*. A smart contract resembles an object in a programming language, with a long-lived *state* recorded in the blockchain. This state is manipulated by a set of *functions* called either directly by clients (top-level calls) or indirectly by other smart contracts (internal calls). To ensure that function calls terminate, each computational step incurs a cost in *gas*, paid by the caller. The caller specifies a maximum amount of gas it is willing to pay, and if the charge exceeds that sum, the computation is terminated and rolled back, and the caller's gas is not refunded. Nevertheless, the rolled-back transaction is still recorded on the blockchain.

A smart contract's code consists of a sequence of bytecode instructions, taken from the Ethereum bytecode instruction set. Every bytecode instruction consumes a certain amount of gas. Each client runs an instance of the Ethereum virtual machine (EVM), which executes calls to smart contracts and runs their sequence of instructions. While users may author smart contracts in higher level languages such as Solidity, these contracts must ultimately be compiled into EVM bytecode, since it is the bytecode that is published in the blockchain. The virtual machine specification, the bytecode instruction set, and all associated gas costs are described in Ethereum's "Yellow Paper" [16].

3.2 Mining and Validation

Smart contracts are first executed by *miners*, or nodes that repeatedly propose new blocks to append to the blockchain. When a miner creates a block, it selects a sequence of client transactions and executes their smart contract codes in sequence, transforming the old contract state into a new state.

Once a block has been appended to the blockchain, that block's smart contracts are re-executed by *validators*, or nodes that reconstruct (and check) the current blockchain state. Each miner validates blocks proposed by other miners, and older blocks are validated by newly-joined miners, or by clients querying the contract state. Once a contract is in a block, it is effectively re-executed forever (in Ethereum), so contract executions by validators vastly exceed executions by miners.

As noted earlier, one drawback of the Ethereum protocol is that a block's contracts are executed in a one-at-a-time order, so miners and validators cannot exploit modern multicore architectures. Contracts cannot be executed concurrently in a naïve way, because they share storage, and may be subject to *data conflicts*, that is, concurrent accesses to the same the storage variables.

In the absence of explicit concurrency guidelines, we execute transactions *speculatively* in parallel, allowing non-conflicting contracts to commit, but rolling back conflicting transactions, and running them sequentially in a second phase.

A novel aspect of this study is that we analyze the effectiveness of speculation against the *historical* record of transactions actually executed on the Ethereum blockchain, replaying their bytecode instructions. We are not aware of an Ethereum virtual machine implementation that supports concurrency, so we simulate concurrent transaction execution by stepping through each transaction's instructions, using eager conflict detection to sort each block's transactions into a conflict-free parallel bin, and a conflicted sequential bin. This strategy is simple and scalable, a natural starting point for an empirical study.

4 The Baseline Experiment

Here we describe the baseline experiment testing the effectiveness of a simple speculative execution strategy against historical transaction data from the Ethereum blockchain. In Section 5, we describe variations on this baseline strategy that probe the sensitivity of our measurements, as well as alternative speculative strategies.

4.1 Setting up the Experiment

We set up an Ethereum node by installing and running the Ethereum Go client, also known as `geth`. The `geth` client allows one to synchronize with other network nodes and reconstruct the blockchain by validating each block. The client comes packaged with an interface for fetching data from the blockchain, as well as debugging tools for inspecting transactions. The debugging API includes a utility for reproducing the bytecode trace of a given transaction.

The `geth` client can run in various synchronization modes, which determine what and how much old state the client records. For example, a client running in *light* mode will keep track of the blockchain's current state only, while in *archive* mode, the client maintains the full log of all previous states.

Using the `geth` API, it is straightforward to retrieve any given smart contract's bytecode. However, the only way to tell how that bytecode affects the blockchain state is by re-executing each transaction before it. Therefore we use `geth`'s tracing utility to re-execute

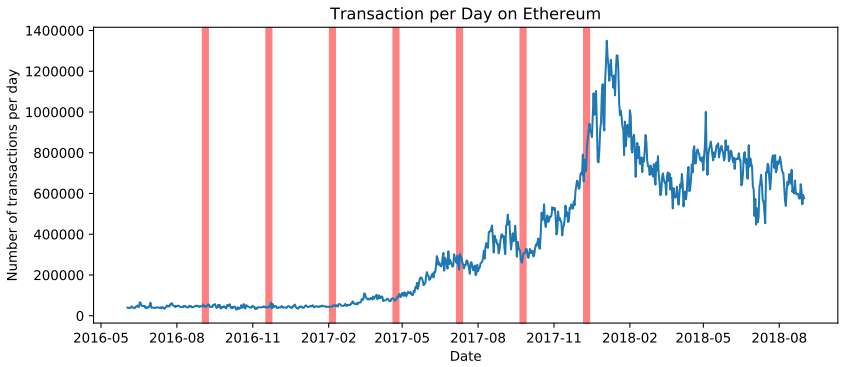


Figure 2 Historical periods analyzed superimposed on number of transactions per day.

Period	Interval	Blocks	Transactions	Calls
1	Sept 1 – 8, 2016	4200	32502	11779
2	Nov 17 – 24, 2016	4200	32648	11070
3	Feb 2 – 9, 2017	3600	26415	10586
4	Apr 20 – 27, 2017	4100	56051	23684
5	July 6 – 13, 2017	3400	197003	84219
6	Sept 21 – 28, 2017	2100	200980	96043
7	Dec 7 – 14, 2017	4100	557471	255872

Figure 3 Number of blocks, transactions, and contract calls analyzed in each historical period.

transactions, running the client in archive mode so it can reference past states. We found that reconstructing the blockchain from scratch in this way was much too slow to be feasible on an ordinary computer (an archive blockchain sync could not even keep up with the current blockchain growth rate), so we obtained a baseline copy of the blockchain from the ConSenSys archive [3] dating from Ethereum's origins in 2016 to early October 2017. Starting from this base, we used the `geth` utilities to synchronize up to December 2017.

Between these dates, the Ethereum blockchain contains about 4 million blocks and 100 million transactions, far too much data to analyze in detail in a reasonable time with reasonable resources. Instead, we chose to focus on seven historical periods between July 2016 (after the DAO fork) and December 2017. Each period spans roughly one week, with consecutive periods separated by 11 weeks, so that each day of the month is considered. Due to computational limitations, we analyze every 10th block in each of the seven historical periods. Figure 2 shows these periods superimposed on the number of transactions per day at that time.

4.2 The Greedy Concurrent EVM

We simulated a concurrent EVM that executes transactions speculatively in parallel using the following *greedy* strategy. For each block, execution proceeds in two phases, an initial *concurrent phase*, and a subsequent *sequential phase*. We consider execution engines with either 16, 32, or 64 threads. In the concurrent phase, each thread chooses a transaction from the block and executes it speculatively. If that transaction encounters a conflict, the transaction's effects are rolled back, and that transaction is deferred to the second, sequential phase. When a thread finishes executing a transaction, it picks another to execute, continuing until all transactions in the current block have been chosen.

The second phase starts when the first phase is complete: the transactions that encountered data conflicts in the first phase are re-executed sequentially. In the second phase, data conflicts are not an issue since transactions are explicitly serialized and state changes committed sequentially.

Two transactions *conflict* if they access the same storage location, and at least one access is a write. Transactions that do not conflict are said to *commute*, because interleaving them in any order yields the same transaction and storage states. At a more granular level, two bytecode operations *conflict* if applying them in different orders yields different storage states. Most bytecode operations, such as arithmetic operations and others that interact with local state, commute with one another. Bytecode operations that interact with shared state, on the other hand, can potentially conflict.

The EVM operations SLOAD and SSTORE read from and write to persistent storage, respectively, and are used for nearly every state access and state modification. By far the most common conflicts arise from conflicting these two operations. Other kinds of conflicts, while possible, are assumed to be too rare to monitor. For example, one transaction might create a new contract, while another calls the newly created contract in a way that creates a race condition: the call may or may not arrive before the contract is initialized. There is also a bytecode operation that reads a given account's balance, which is part of the blockchain's shared state.

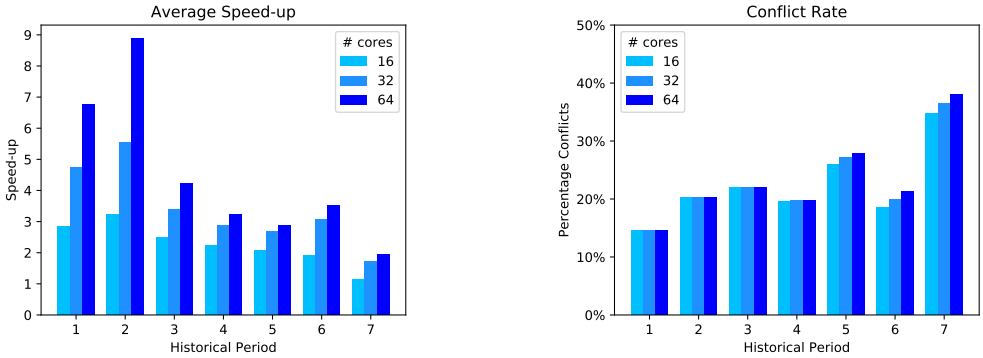
We detect conflicts by associating a *read-write lock* with each storage location. Each SLOAD (respectively, SSTORE) operation requests that location's lock in read (write) mode. If a transaction requests a lock that is already held in a conflicting mode by another transaction, the requesting transaction is rolled back and deferred to the next phase. No locks are released until the concurrent phase ends, even those held by aborted transactions. This is to ensure that no interleaving of transactions in the concurrent bin result in a conflict when re-executed by validators.

4.3 Sampling and Evaluation

In this section, we evaluate the concurrent speculative execution strategy on the data set of transaction traces as described in the experimental setup (see Section 4.1). Traces from the historical Ethereum blockchain are used to simulate what would have happened if the original blocks of transactions were instead re-executed concurrently.

Our principal figure of merit is *speed-up*: the ratio between the time to execute a block's transactions sequentially, and the time to execute the same transactions concurrently and speculatively. For now, we use cumulative gas costs as a proxy for time. Then to estimate speed-up, we measure the ratio between (1) the gas consumed to execute a block's transactions sequentially, versus (2) the maximum gas used by any thread in the parallel phase plus the gas needed to execute the sequential phase. Note that an aborted transaction may be counted twice: once (partially) in the concurrent phase, up to when the transaction is aborted, and once in the sequential phase when the aborted transaction is re-executed in isolation.

4.4 Baseline Results



(a) Average speed-up.

(b) Average conflict rate.

Figure 4 Average speed-up and conflict rate for each of the seven historical time intervals. Increased Ethereum activity over time is reflected by decreasing speed-up and higher conflict rates.

Figure 4 summarizes speed-up statistics over the seven historical periods, along with the *conflict rates*, or the percentage of contract calls that abort per block. The average speed-up and conflict rate are shown for simulated VMs of 16, 32, and 64 cores, where averages are weighted by the number of contract call transactions in each block.

Earlier periods display higher speed-ups and lower conflict rates because transaction volume and contention are low. For example, speed-up is as high as 3.23 in the second historical period on a simulated EVM with 16 cores, and this number rises to 8.87 for a 64 core EVM. During the same interval, contract calls abort at a rate of only 20%.

As the volume of transactions increases over time, however, so does the rate of transaction conflict, so more time is spent sequentially re-executing transactions aborted during the concurrent phase. This naturally leads to lower speed-ups, since there is less opportunity to parallelize transaction execution. Indeed, during the December 2017 period, with 16 threads, roughly 34% of transactions abort. Nevertheless, it is notable that there is still a modest but positive speed-up of 1.13 even then. Moreover, this speed-up effectively doubles, to slightly more than 2 when there are 64 threads. When transaction volume is higher, using more threads yields more speed-up.

4.5 Speed-up Distributions

The average speed-up of each historical period provides little insight into how the blocks' speed-ups are distributed in each period. Here, we further analyze the performance of speculative execution by looking at the distributions of these speed-ups. Due to space limitations, we focus on historical periods 5, 6, and 7, since these have the highest transaction volumes.

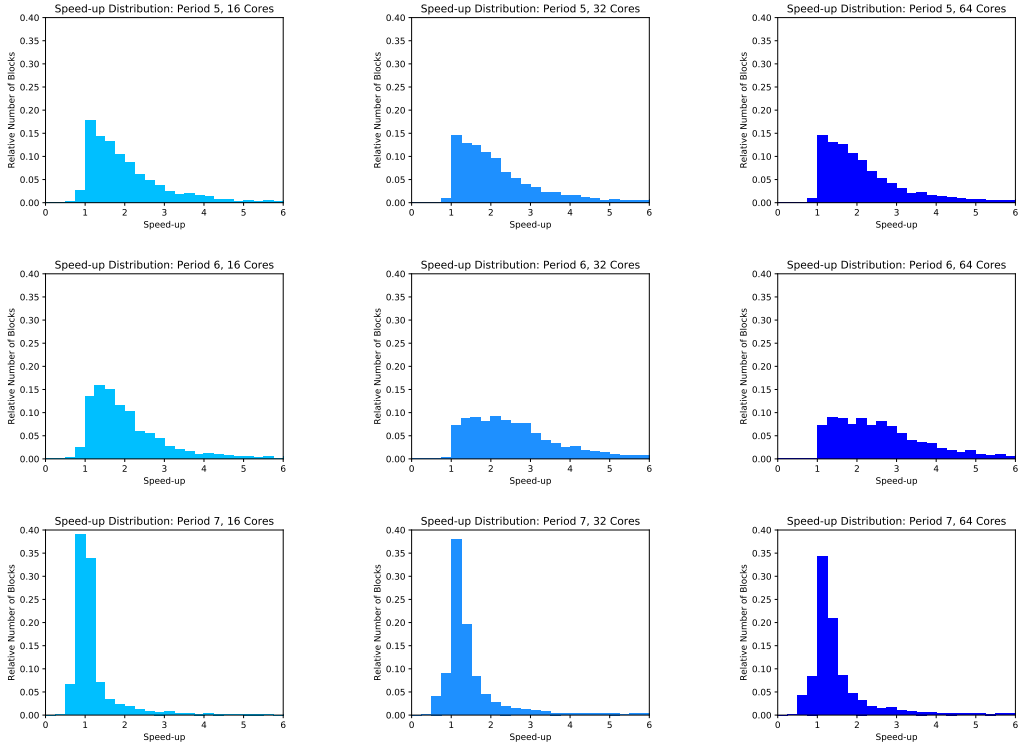


Figure 5 Speed-up distributions for each combination of historical periods 5, 6, and 7, and with 16, 32, and 64 cores. Each plot depicts one such combination. Blocks are binned in increments of 0.25 according to speed-up realized. Distributions are normalized so that total area is roughly 1.

Speed-up distributions of the three periods are depicted in Figure 5. Each plot is normalized to represent an approximate probability density, so that the y-axis is interpreted as a relative count. Since there is relatively less contention in Periods 5 and 6, their speed-up distributions are flatter, with longer tails. This is because more blocks during these periods are able to exhibit a higher amount of speed-up. However, when the transaction rate is higher, as in seventh period of December 2017, the distributions are concentrated closed to 1, since more work is wasted as the number of aborted transactions rises.

As noted before, some blocks may realize a slowdown when using our concurrent execution strategy, a result of the cost of aborting and re-executing conflicting transactions. The relative number of such blocks is shown in plots of Figure 5, where regions to the left of 1 represent blocks that are slowed down. Making more cores available to the EVM appears to reduce the number of slowed-down blocks for each of the three historical periods. Though fewer than 3% blocks are slowed down for both Periods 5 and 6 when using 16 cores, this percentage jumps to 46% for Period 7. However, when the EVM's number of cores is increased to 32, the percentage of blocks that are slowed down drops substantially to only 13%.

4.6 Storage Hot-Spots

Next, we investigate transaction contention by analyzing how often certain storage addresses are accessed by transactions. In particular, we look at memory addresses that are conflict points for pairs of transactions, and how many times each such address results in a conflict. Addresses that attract a high number of conflicts are informally called *hot-spots*.

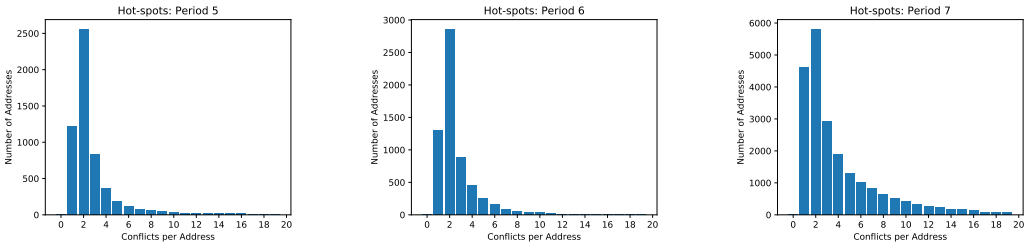


Figure 6 Histograms of conflicts per address. Addresses that results in a large number of conflicts are hot-spots. Period 7 has a much heavier tail, corresponding to many more address hot-spots.

In Figure 6, we plot histograms illustrating the number of conflicts per address, for historical periods 5, 6, and 7. Each storage address with at least one conflict is binned according to how many conflicts occurred at that address. For example, in Period 5, there are 2562 unique storage addresses at which there were exactly 2 conflicts. Periods 5 and 6 have relatively few storage addresses with high contention, and this pattern is similar to the earlier historical periods. However, Period 7's histogram has a much heavier tail than the other two histograms, meaning that there are many more storage addresses with larger numbers of conflicts. In other words, there is a handful of addresses that many different transactions attempt to access, most likely a result of so many transactions calling the same few contracts. We elaborate on this observations in Section 5.6 below.

5 Alternative Experiments

We extended the baseline experiment with a number of other experiments intended to test the effectiveness of alternative strategies, and to test the sensitivity of our approximations.

5.1 Sampling

To test whether sampling 1 in 10 blocks during a period yielded a distorted view, we ran a detailed simulation of one period (Period 5) to compare the results with the sampled simulation. There is a modest difference in speed-up, and an even more negligible difference in abort rate. Some discrepancy can also be explained by the inherent randomness of a concurrent scheduler.

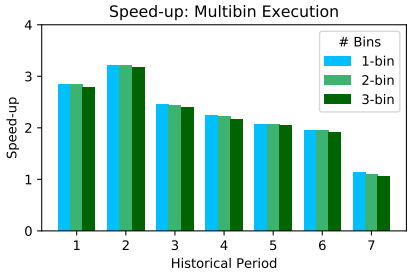
Sampling rate	**Speed-up**			**Abort rate**		
	16 cores	**32 cores**	**64 cores**	**16 cores**	**32 cores**	**64 cores**
1-in-10 sampling	2.063	2.694	2.859	25.98%	27.23%	27.88%
All blocks	2.085	2.717	2.913	25.96%	27.21%	27.86%

Figure 7 Accuracy of 1-in-10 sampling for historical period 5.

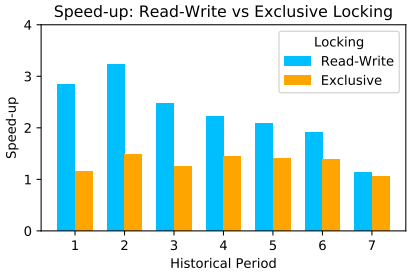
5.2 Multiple Phases

The greedy, two-phase strategy can be generalized to encompass multiple concurrent phases, so that each transaction that was deferred in one concurrent phase is instead re-executed in another subsequent concurrent phase. It is possible that multiple concurrent phases could provide additional speed-up. However, as illustrated in Figure 8a, we found in our

experiments that executing two concurrent phases almost always yields less speed-up than executing a single concurrent phase. This decrease is due to the duplicate work performed by transactions rolled back in multiple phases, with not enough additional speed-up yielded in the latter concurrent phases. Therefore, in practice, one current phase is sufficient to realize almost all potential concurrency.



(a) Speed-up with multiple concurrent phases for transaction execution.



(b) Speed-up when the EVM uses mutex locks to lock storage addresses.

Figure 8 Multiple phases and data conflicts figures.

5.3 Data Conflicts

In our simulated concurrent EVM, transactions access storage addresses by first acquiring a read-write lock. This allows multiple transactions to read an address, with no conflict, if there are no concurrent writers. In principle, this decreases the number of conflicts when compared to using mutex locks, hence increasing the potential speed-up.

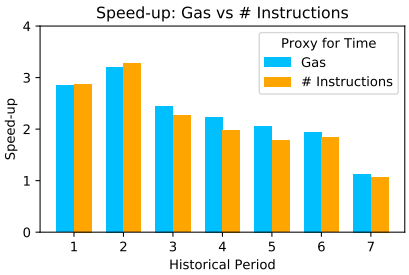
To determine whether read-write locks reduce data conflicts in practice, we investigated the effect of simplifying the conflict model by merging all data accesses into a single conflict set, by using mutex locks for conflict detection. With the exception of the last historical period in which speed-up is already low, the speed-ups were substantially worse than the speed-ups obtained by distinguishing between read and write accesses. See Figure 8b for a comparison of the two locking schemes when using 16 cores. These results suggest that there is significant value in implementing a concurrent EVM with read-write locks, instead of the simpler mutex locks.

5.4 Proxies for Time

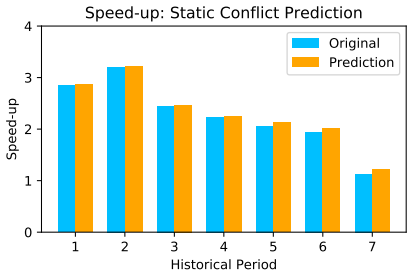
Since we do not have access to an actual concurrent EVM, we must estimate how long it takes to execute each transaction. There are two straightforward choices: we can count the number of instructions executed by each transaction, or we can tally the gas cost of executing the transaction's instructions. The first choice assumes that each EVM instruction takes roughly the same time to execute, while the second assumes that instruction gas cost is roughly proportional to execution time. For example, the arithmetic operations MUL and DIV require 5 units of gas, while SSTORE and SLOAD cost 20000[1] and 200 units respectively.

All speed-ups reported so far were measured in terms of gas costs. As a sanity check, for 16 cores, we recomputed speed-ups using instruction count as a proxy for time. These speed-ups are shown in Figure 9a. There does not appear to be any significant qualitative difference between gas cost and instruction count.

[1] SSTORE costs 20000 units when storing a non-zero value, but costs only 5000 units otherwise.



(a) Speed-up when instruction count is used to measure transaction execution time.



(b) Speed-up under static conflict prediction. Aborted transactions costs effectively ignored.

Figure 9 Proxies for time and static conflict prediction figures.

5.5 Static Conflict Prediction

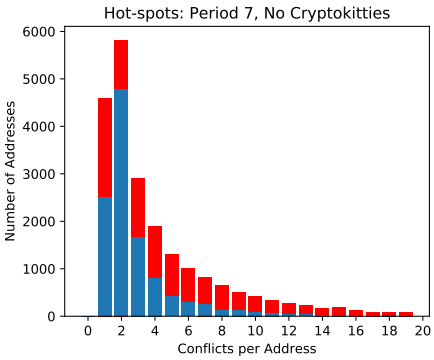
If we were able to predict whether a transaction would abort if executed speculatively, then we could save the cost of transaction roll-back and retry. We can simulate the effect of a perfect conflict predictor simply by ignoring the cost of aborted transactions, as in Figure 9b. However, doing so yields a negligible change in speed-ups, less than 0.1% in most cases. The only exception is the very last period (December 2017), where contention was very high. For that historical period, the average speed-up increases from 1.13 to 1.22. These numbers suggest that static conflict analysis, if accurate enough, may yield modest gains during periods of high contention.

5.6 Omitting Hot-spot Contracts

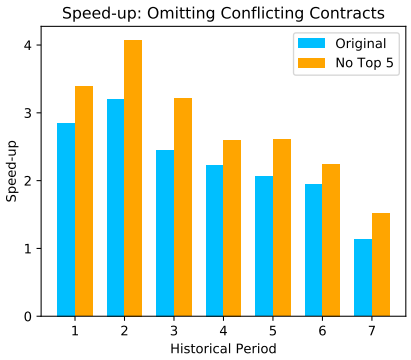
In most of the previous analysis, Period 7 stands out among the selected historical periods as being particularly high volume and contentious. Recall that this period was sampled from December 2017, which is close to peak Ethereum transaction activity. More specifically, Period 7 occurred when there was great interest from the general public over *CryptoKitties* [2] [6], a recreational game deployed on Ethereum in which users create, breed, and trade virtual cats. The popularity of CryptoKitties is especially apparent when using any blockchain explorer to browse Ethereum transactions during December 2017. The popularity of CryptoKitties was responsible for congesting the Ethereum network, though interest in it has since died down.

In light of the CryptoKitties frenzy, we reanalyzed Period 7 under a hypothetical scenario in which the CryptoKitties contracts [2] did not exist. This is easily accomplished by ignoring all calls to the CryptoKitties contracts when tracing each block's transactions and replaying them, and calculating the resulting statistics.

[2] The vast majority of calls to CryptoKitties are to its core contract (which has address hash `0x06012c8cf97bead5deae237070f9587f8e7a266d`), and to an auction contract (which has address hash `0xb1690c08e213a35ed9bab7b318de14420fb57d8c`).



(a) The effect on hot-spot counts when CryptoKitties contract calls are ignored.



(b) Speed-up when calls to the five most conflicting contracts are ignored.

Figure 10 Speed-up analysis after removing high-contention or high-volume contracts.

After filtering out all calls to CryptoKitties contracts, speed-up using 16 cores rises from 1.13 to 1.65. These calls accounted for about 31% of all contract calls, and furthermore, by filtering out CryptoKitties, the number of contract calls per block drops from 62 to 43, which is much closer to the 46 calls per block in Period 6. While a speed-up of 1.65 does not quite match speed-up from older periods, it is still clear from this simple analysis that much of the contention in Period 7 is caused solely by CryptoKitties.

To further illustrate the impact that CryptoKitties had, we reproduce the conflict histogram from the previous section, using the same hypothetical scenario with no CryptoKitties. Recall that the conflict histogram of Period 7 had a much heavier tail than the corresponding histograms of older intervals. However when CryptoKitties is removed, Period 7's histogram (Figure 10a) has a much thinner tail, resembling the other histograms. Indeed, in Period 6, 10% of conflicting storage addresses have at least 5 conflicts, but in Period 7, this same number is 30%. However, if CryptoKitties contracts are ignored, then only 14% or conflicting address have at least 5 conflicts. These numbers demonstrate that CryptoKitties is responsible for many storage hot-spots.

We generalize this analysis to the other historical periods by determining the top five most conflicting smart contracts in each period, and reproducing scenarios where none of the highly conflicting contracts were called. As shown in Figure 10b, this results in a noticeable speed-up in each period, not just the 7th. Most of these highly conflicting contracts are actually token contracts; in fact, the most contentious contract from each period is either a token (or a token exchange) contract. Therefore, analyzing these small sets of contracts may provide insight into how to reduce contention when speculatively executing smart contracts in parallel.

6 Discussion

As noted, this study is exploratory, incorporating various approximations and omissions. Most such omissions are the result of the absence of a standard concurrent EVM implementation. This study does not account for some EVM overheads, such as the costs for *value transfers*, where one account transfers ether directly to another, without modifying storage. Value transfers typically commute, and are likely much faster to execute than contract calls, though they do make up the majority of all Ethereum transactions. Other sources of overhead include solving a cryptographic puzzle to compute proof of work, which would affect miners' speed-up but not the validators'.

In the absence of a timing model for a concurrent EVM, this study uses gas costs (or instruction counts) as proxies for time when computing speed-up. As noted, both proxies yield essentially equivalent results.

This study relies on sampled blocks because the volume of data in the Ethereum blockchain is simply too large to make exhaustive analysis practical or rewarding. An archive synchronization of the blockchain was a major computational overhead for this study, in addition to recovering transaction traces. These overheads may be reduced as further Ethereum tools and utilities are developed.

There are several reasons why speculative parallelism may sometimes yield little, or even negative speed-up. For example, a block might contain one transaction substantially longer than the others, whose execution time dominates the block execution time. In this case, it is impossible to achieve much speed-up, no matter how these transactions are scheduled and distributed among multiple cores. Or if a block contains very few contract call transactions, there is little opportunity for speed-up. If a block's transactions all access the same storage location, perhaps because they all access the same popular contract [6], then speculation will produce a negative speed-up as a result of the cost of rolling back so many misspeculated transactions.

7 Conclusions

Our results suggest that a simple speculative strategy based on read-write set overlap can produce non-trivial speed-ups, but that such speed-ups will decline as transaction rates and conflict rates increase. More aggressive strategies, such as adding additional parallel phases, seem to provide little additional benefit, because conflict appears to be bursty: if one transaction conflicts with another, then it probably conflicts with multiple others.

The results of this study suggest that the most promising way to further increase parallelism in Ethereum-style smart contract execution is to reduce the conflict rate, perhaps by focusing on reducing unnecessary conflicts. We observed that splitting transactions' data sets into read sets and write sets decreased conflict rates substantially, suggesting that conflict rates are sensitive to the *semantics* of concurrent operations on shared data. This observation suggests that conflict rates might be reduced even further if the execution engine could do a better job of recognizing when operations commute at the semantic level. For example, transactions that increment or decrement the same account balance (a common occurrence) have overlapping read and write sets, and are therefore deemed to conflict. At the semantic level, however, these operations commute (in the absence of overflow or underflow), so as long as the virtual machine's memory operations are atomic, those operations need not conflict. (Our study could not detect which conflicts are real, and which are artifacts, because only compiled bytecode was available for analysis).

It might be profitable to investigate the effects of endowing the virtual machine with intrinsic data types such as atomic counters or atomic sets that provide many commuting mutator operations. Studying highly conflicting token contracts may provide insight into which kinds of data types or operations would best alleviate contention.

Periods of high contention and low speed-up are caused by a relatively small number of popular contracts. Currently, smart contract designers have no guidance how to avoid speculative data conflicts, nor any incentive to do so. Our results suggest that there is a need to devise incentives for smart contract programmers to design contracts in ways that reduce conflicts, either by eliminating spurious conflicts, or by exploiting improved commuting bytecode instructions.

References

1. Parwat Singh Anjana, Sweta Kumari, Sathya Peri, Sachin Rathor, and Archit Somani. An Efficient Framework for Concurrent Execution of Smart Contracts. *CoRR*, abs/1809.01326, 2018. arXiv:1809.01326.
2. Nellie Bowles. CryptoKitties, Explained... Mostly. *The New York Times*, December 2017. URL: https://www.nytimes.com/2017/12/28/style/cryptokitties-want-a-blockchain-snuggle.html.
3. Consensys. Harness the power of Ethereum. https://consensys.net/. As of 22 November 2018.
4. Hyperledger Consortium. HyperLedger Fabric Release 1.3. URL: https://hyperledger-fabric.readthedocs.io/en/release-1.3/.
5. R3 Corda. Writing a contract. URL: https://docs.corda.net/releases/release-M1.0/tutorial-contract.html. As of 31 December 2018.
6. Crytokitties. CryptoKitties | Collect and breed digital cats! https://www.cryptokitties.co/. As of 22 November 2018.
7. Thomas Dickerson, Paul Gazzillo, Maurice Herlihy, and Eric Koskinen. Adding Concurrency to Smart Contracts. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, PODC '17, pages 303–312, New York, NY, USA, 2017. ACM. doi:10.1145/3087801.3087835.
8. Ethereum. https://github.com/ethereum/.
9. IOHK. Cardano Settlement Layer Documentation. https://cardanodocs.com/technical/plutus/introduction/. As of 31 December 2018.
10. Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol, 2017. As of 24 November 2018 URL: https://iohk.io/research/papers/#ouroboros-a-provably-secure-proof-of-stake-blockchain-protocol.
11. David Mazieres. The Stellar Consensus Protocol: A Federated Model for Internet-level Consensus, 2015. As of 24 November 2018. URL: https://www.stellar.org/papers/stellar-consensus-protocol.pdf.
12. Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System, May 2009. URL: http://www.bitcoin.org/bitcoin.pdf.
13. David Schwartz, Noah Youngs, and Arthur Britto. The Ripple Protocol Consensus Algorithm. https://ripple.com/files/ripple_consensus_whitepaper.pdf, 2014.
14. Solidity documentation. http://solidity.readthedocs.io/en/latest/index.html.
15. Nick Szabo. Formalizing and Securing Relationships on Public Networks. *First Monday*, 2(9), 1997. URL: http://firstmonday.org/ojs/index.php/fm/article/view/548.
16. Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151:1–32, 2014.

Atomic Appends: Selling Cars and Coordinating Armies with Multiple Distributed Ledgers

Antonio Fernández Anta
IMDEA Networks Institute, Madrid, Spain
antonio.fernandez@imdea.org

Chryssis Georgiou
Dept. of Computer Science, University of Cyprus, Nicosia, Cyprus
chryssis@cs.ucy.ac.cy

Nicolas Nicolaou
Algolysis Ltd, Cyprus
nicolas@algolysis.com

Abstract

The various applications using Distributed Ledger Technologies (DLT) or blockchains, have led to the introduction of a new "marketplace" where multiple types of digital assets may be exchanged. As each blockchain is designed to support specific types of assets and transactions, and no blockchain will prevail, the need to perform *interblockchain* transactions is already pressing.

In this work we examine the fundamental problem of interoperable and interconnected blockchains. In particular, we begin by introducing the *Multi-Distributed Ledger Objects* (MDLO), which is the result of aggregating multiple *Distributed Ledger Objects* – DLO (a DLO is a formalization of the blockchain) and that supports append and get operations of records (e.g., transactions) in them from multiple clients concurrently. Next we define the *AtomicAppends* problem, which emerges when the exchange of digital assets between multiple clients may involve appending records in more than one DLO. Specifically, AtomicAppend requires that either *all* records will be appended on the involved DLOs or *none*. We examine the solvability of this problem assuming *rational and risk-averse* clients that may *fail by crashing*, and under different client *utility* and *append* models, *timing models*, and client *failure scenarios*. We show that for some cases the existence of an intermediary is *necessary* for the problem solution. We propose the implementation of such intermediary over a specialized blockchain, we term *Smart DLO* (SDLO), and we show how this can be used to solve the AtomicAppends problem even in an asynchronous, client competitive environment, where all the clients may crash.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases DLO, Interoperability, Atomic Appends, Rational Clients, Fault-tolerance

Digital Object Identifier 10.4230/OASIcs.Tokenomics.2019.5

Funding Partially supported by the Spanish Ministry of Science, Innovation and Universities grant DiscoEdge (TIN2017-88749-R), the Comunidad de Madrid grant EdgeData-CM (P2018/TCS-4499), the NSF of China grant 61520106005, and research funds from the University of Cyprus (CG-RA2019).

Acknowledgements We would like to thank Kishori Konwar, Michel Raynal, and Gregory Chockler for insightful discussions.

1 Introduction

Blockchain systems, cryptocurrencies, and distributed ledger technology (DLT) in general, are becoming very popular and are expected to have a high impact in multiple aspects of our everyday life. In fact, there is a growing number of applications that use DLT to support their operations [26]. However, there are many different blockchain systems, and new ones are

proposed almost everyday. Hence, it is extremely unlikely that one single DLT or blockchain system will prevail. This is forcing the DLT community to accept that it is inevitable to come up with ways to make blockchains interconnect and interoperate.

The work in [7] proposed a formal definition of a reliable concurrent object, termed Distributed Ledger Object (DLO), which tries to convey the essential elements of blockchains. In particular, a DLO is a sequence of records, and has only two operations, append and get. The append operation is used to attach a new record at the end of the sequence, while the get operation returns the sequence.

In this work we initiate the study of systems formed by multiple DLOs that interact among each other. To do so, we define a basic problem involving two DLOs, that we call *the Atomic Append problem*. In this problem, two clients want to append new records in two DLOs, so that either both records are appended or none. The clients are assumed to be selfish, but rational and risk-averse [21], and may have different incentives for the different outcomes. Additionally, we assume that they may fail by crashing, which makes solving the problem more challenging. We observe that the problem cannot be solved in some system models and propose algorithms that solve it in others.

1.1 Related Work

The Atomic Append problem we describe above is very related to the multi-party fair exchange problem [8], in which several parties exchange commodities so that everyone gives an item away and receives an item in return. The proposed solutions for this problem rely on cryptographic techniques [17, 19] and are not designed for distributed ledgers. In this paper, as much as possible, we want to solve Atomic Appends on DLOs via their two operations append and get, without having to rely on cryptography or smart contracts.

Among the first problems identified involving the interconnection of blockchains was Atomic Cross-chain Swaps [13], which can also be seen as a version of the fair exchange problem. In this case, two or more users want to exchange assets (usually cryptocurrency) in multiple blockchains. This problem can be solved by using escrows, hashlocks and timelocks: all assets are put in escrow until a value x with a special hash $y = hash(x)$ is revealed or a certain time has passed. Only one of the users knows x, but as soon as she reveals it to claim her assets, everyone can use it to claim theirs. Observe that this solution assumes synchrony in the system, in the sense that timelocks assume that the time to claim an asset is bounded and known, and that timeouts can be used to detect crashes.

This technique was originally proposed in on-line fora for two users [1], and it has been specified, validated, adapted, and used [20, 25]. For instance, the Interledger system [11] will use a generalization of atomic swaps to transfer (and exchange) currency in a network of blockchains and connectors, allowing any client of the system to interact with any other client. The Lightning network [18, 22] also allows transfers between any two clients via a network of micro-payment channels using a generalized atomic swap. Both Interledger and Lighting route and create one-to-one transfer paths in their respective networks. Herlihy [13] has formalized and generalized atomic cross-chain swaps beyond one-to-one paths, and shows how multiple cross-chain swaps can be achieved if the transfers form a strongly connected directed graph. Herlihy proves that the best strategy, in Game Theoretic sense, for the users is to follow the proposed algorithm, and that someone that follows it will never end up worst than at the start.

Unlike in most blockchain systems, in Hyperledger Fabric [5, 6] it is possible to have transactions that span several blockchains (blockchains are called *channels* in Hyperledger Fabric). This allows solving the atomic cross-chain swap problem using a third trusted

channel or a mechanism similar to a two-phase commit [5]. Additionally, these solutions do not require synchrony from the system. The ability of channels to access each other's state and interact is a very interesting feature of Hyperledger Fabric, very in line with the techniques we assume from advanced distributed ledgers in this paper. Unfortunately, they seem to be limited to the channels of a given Hyperledger Fabric deployment.

There are other blockchain systems under development that, like Hyperledger Fabric, will allow interactions between the different chains, presumably with many more operations than atomic swaps. Examples are Cosmos [2] or PolkaDot [4]. These systems will have their own multi-chain technology, so only chains in a given deployment can initially interact, and other blockchain will be connected via gateways. Another proposal for interconnection of blockchains is Tradecoin [12], whose target is to interconnect all blockchains by means of gateways, trying to reproduce the way Internet works. Since the gateways will be clients of the blockchains, the functionality of the global interledger system will be limited by what can be done from the edge of the blockchains (i.e., by the blockchains' clients).

The practical need of blockchain systems to access the outside world to retrieve data (e.g., exchange rates, bank account balances) has been solved with the use of *blockchain oracles*. These are relatively reliable sources of data that can be used inside a blockchain, typically in a smart contract. The weakest aspect of blockchain oracles is trust, since the outcome or actions of a smart contract will be as reliable as the data provided by the oracle. As of now, it seems there is no good solution for this trust problem, and blokchains have to rely on oracle services like Oraclize [3].

1.2 Contributions

As mentioned above, in this paper we extend the study of the distributed ledger reliable concurrent object DLO started in [7] to systems formed of several such objects. Hence, the first contribution is the definition of the Multiple DLO (MDLO) system, as the aggregation of several DLOs (in similar way as a Distributed Shared Memory is the aggregation of multiple registers [24]). The second contribution is the definition of a simple basic problem in MDLO systems: the *2-AtomicAppends problem.* In this problem, the objective is that two records belonging to two different clients are appended to two different DLOs atomically. Hence, either both records are appended or none is. Of course, this problem can be generalized in a natural way to the k-Atomic Appends problem, involving k clients with k records and up to k DLOs.

Another contribution, in our view, is the introduction of a crash-prone risk-averse rational client model, which we believe is natural and practical, especially in the context of blockchains. In this model, clients act selfishly trying to maximize their utility, but minimizing the risk of reducing it. We consider that this behavior is not a failure, but the nature of the client, and any algorithm proposed under this model (e.g., to solve the 2-AtomicAppends problem) must guarantee that clients will follow it, because their utility will be maximized without any risk. For a complete specification of the clients' rationality their utility function has to be provided. Two utility models are proposed. In the *collaborative utility model*, both clients want the records to be appended over any other alternative. In the *competitive utility model* a client still wants both records appended, but she prefers that only the other client appends. This client model is complemented with the possibility that clients can fail by crashing.

We explore hence the solvability of 2-AtomicAppends in MDLO systems in which the DLOs are reliable but may be asynchronous, and the clients are rational but may fail by crashing. The first results we present consider a system model in which clients do not crash, and show that Collaborative 2-AtomicAppends can be solved even under asynchrony, while

Competitive 2-AtomicAppends cannot be solved. Then, we further study Collaborative 2-AtomicAppends if clients can crash. In the case that at most one of the two clients can crash, we show that, if each client must append its own record (what we call *no delegation*), Collaborative 2-AtomicAppends cannot be solved even under synchrony. This justifies exploring the possibility of *delegation:* any client can append any record, if she knows it. We show that in this case Collaborative 2-AtomicAppends can be solved, even if the system is asynchronous (termination is only guaranteed under synchrony, though). However, delegation is not enough if both clients can crash, even under synchrony. (See Table 2 for an overview.)

The negative results (for Competitive 2-AtomicAppends even without crash failures and for Collaborative 2-AtomicAppends with up to 2 crashes) justifies exploring alternatives to appending directly or delegating among clients. Hence, we propose the use of an entity, external to the clients, that coordinates the appends of the two records. In fact, this entity is a special DLO with some level of intelligence, which we hence call *Smart DLO* (SDLO). The SDLO is by design a reliable entity to which clients can delegate (via appending in the SDLO) the responsibility of appending their records to their respective DLOs when convenient. The SDLO hence collects all the records from the clients and appends them. Since the SDLO is reliable, all the appends will complete. If some record is missing, the SDLO issues no append, to guarantee the properties of the 2-AtomicAppends problem. Thus, the SDLO can be used to solve Competitive and Collaborative k-AtomicAppends even when all clients can crash.

We believe that SDLO opens the door to a new type of interconnection and interoperability among DLOs and blockchains. While the use of oracles to access external information in a smart contract (maybe from another blockchain) is widely known, we are not familiar with blockchain systems in which one blochchain (i.e., possibly a smart contract) issues transactions in another blockchain. We believe this is a concept worth to be explored further.

The rest of the paper is structured as follows. The next section describes the model used and defines the AtomicAppends problem. Section 3 explores the 2-AtomicAppends problem when clients cannot crash. Section 4 studies the 2-AtomicAppends problem when clients can crash but SDLOs are not used. Section 5 introduces the SDLO and shows how it solves the AtomicAppends problem. Finally, Section 6 presents conclusions and future work.

2 Problem Statements and Model of Computation

2.1 Objects and Histories

An object type T is defined over the domain of values that any object of type T may take, and the operations that any object of type T supports. An object O of type T is a *concurrent object* if it is a shared object accessed by multiple processes [23]. A *history* of operations on an object O, denoted by H_O, is the sequence of operations invoked on O. Each operation π contains an *invocation* and a matching *response* event. Therefore, a *history* is a sequence of invocation and response events, starting with an invocation. We say that an operation π is *complete* in a history H_O, if the history contains both the invocation and the *matching* response events of π. History H_O is *complete* if it only contains complete operations. History H_O is *well-formed* if no two invocation events that do not have a matching response event in H_O belong to the same process p. That is, each process p invokes one operation at a time. An object history H_O is *sequential*, if it contains a sequence of alternating invocation and matching response events, starting with an invocation and ending with a response. We say that an operation π_1 *happens before* an operation π_2 in a history H_O, denoted by $\pi_1 \rightarrow \pi_2$, if the response event of π_1 appears before the invocation event of π_2 in H_O.

The Ledger Object (LO). A *ledger* $\mathcal{L}$ (as defined in [7]) is a concurrent object that stores a totally ordered sequence $\mathcal{L}.S$ of *records* and supports two operations (available to any process p): (i) $\mathcal{L}.\mathsf{get}_p()$, and (ii) $\mathcal{L}.\mathsf{append}_p(r)$. A *record* is a triple $r = \langle \tau, p, v \rangle$, where p is the identifier of the process that created record r, τ is a *unique* record identifier from a set $\mathcal{T}$, and v is the data of the record drawn from an alphabet Σ. We will use $r.p$ to denote the id of the process that created record r; similarly we define $r.\tau$ and $r.v$. A process p invokes an $\mathcal{L}.\mathsf{get}_p()$ operation to obtain the sequence $\mathcal{L}.S$ of records stored in the ledger object $\mathcal{L}$, and p invokes an $\mathcal{L}.\mathsf{append}_p(r)$ operation to extend $\mathcal{L}.S$ with a new record r. Initially, the sequence $\mathcal{L}.S$ is empty.

▶ **Definition 1** (Sequential Specification of a LO [7]). *The* sequential specification *of a ledger $\mathcal{L}$ over the sequential history $H_{\mathcal{L}}$ is defined as follows. The value of the sequence $\mathcal{L}.S$ of the ledger is initially the empty sequence. If at the invocation event of an operation π in $H_{\mathcal{L}}$ the value of the sequence in ledger $\mathcal{L}$ is $\mathcal{L}.S = V$, then:*

1. *if π is an $\mathcal{L}.\mathsf{get}_p()$ operation, then the response event of π returns V, while the value of $\mathcal{L}.S$ does not change, and*
2. *if π is an $\mathcal{L}.\mathsf{append}_p(r)$ operation (and $r \notin V$), then at the response event of π the value of the sequence in ledger $\mathcal{L}$ is $\mathcal{L}.S = V \| r$ (where $\|$ is the concatenation operator).*

In this paper we assume that ledgers are *idempotent*, therefore a record r appears only once in the ledger even when the same record r is appended to the ledger by multiple append operations (and hence the $r \notin V$ in the definition above).

2.2 Distributed Ledger Objects (DLO) and Multiple DLOs (MDLO)

Distributed Ledger Objects (DLO). A *Distributed Ledger Object (DLO)* $\mathcal{DL}$, is a concurrent LO that is *implemented* by (and possibly replicated among) a set $\mathcal{S}$ of (possibly distinct and geographically dispersed) computing devices, we refer as *servers*. Like any LO, $\mathcal{DL}$ supports the operations get() and append(). We refer to the processes that invoke the get() and append() operations on $\mathcal{DL}$ as *clients*.

Each server $s \in \mathcal{S}$ may fail. Thus, the distribution and replication of $\mathcal{DL}$ offers availability and survivability of the ledger in case a subset of servers fail. At the same time, the fact that multiple clients invoke append() and get() requests to different servers, raises the challenge of *consistency*: what is the latest value of the ledger when multiple clients access the ledger concurrently? The work in [7] defined three consistency semantics to explain the behavior of append() and get() operations when those are invoked concurrently by multiple clients on a single DLO. In particular, they defined *linearizable* [14, 16], *sequential* [15], and *eventual* [9] consistent DLOs. In this work we will focus on *linerizable* DLOs which according to [7] are defined as follows:

▶ **Definition 2** (Linearizable Distributed Ledger Object [7]). *A distributed ledger $\mathcal{DL}$ is* linearizable *if, given any complete, well-formed history $H_{\mathcal{DL}}$, there exists a sequential permutation σ of the operations in $H_{\mathcal{DL}}$ such that:*

1. *σ follows the sequential specification of a ledger object (Definition 1), and*
2. *for every pair of operations π_1, π_2, if $\pi_1 \to \pi_2$ in $H_{\mathcal{DL}}$, then π_1 appears before π_2 in σ.*

Multiple DLOs (MDLO). A *Multi-Distributed Ledger Object* $\mathcal{MDL}$, termed MDLO, consists of a collection D of (heterogeneous) DLOs and supports the following operations: (i) $\mathcal{MDL}.\mathsf{get}_p(\mathcal{DL})$, and (ii) $\mathcal{MDL}.\mathsf{append}_p(\mathcal{DL}, r)$. The get returns the sequence of records $\mathcal{DL}.S$, where $\mathcal{DL} \in D$. Similarly, the append operation appends the record r to the end of the sequence $\mathcal{DL}.S$, where $\mathcal{DL} \in D$. From the locality property of linearizability [14] it follows that a MDLO is linearizable, if it is composed of linearizable DLOs. More formally:

▶ **Definition 3** (Linearizable Multi-Distributed Ledger Object)**.** *A multi-distributed ledger $\mathcal{MDL}$ is* linearizable *if $\forall \mathcal{DL} \in D$, $\mathcal{DL}$ is linearizable, where D is the set of DLOs $\mathcal{MDL}$ contains.*

For the rest of this paper, unless otherwise stated, we will focus on MDLOs consisting of two DLOs. The same techniques can be generalized in MDLOs with more than two DLOs. In particular, we consider the records of two clients, A and B, on two different DLOs. For convenience we use DLO_X to denote the DLO appended by records from X, for $X \in \{A, B\}$. Similarly we denote as r_X the record that $X \in \{A, B\}$ wants to append on DLO_X. Furthermore, we view the DLOs and MDLOs as black boxes that reliably implement the specified service, without going into further implementation details.

2.3 AtomicAppends: Problem Definition

Multi-DLOs allow clients to interact with different DLOs concurrently. This is safe when the records involved in concurrent operations are independent. However, it may raise semantic consistency issues when there exists inter-dependent records, e.g. a record r_A must be inserted in DLO_A when a record r_B is inserted in DLO_B and vice versa. More formally, we say that a record r *depends* on a record r', if r may be appended on its intended DLO, say $\mathcal{DL}$, only if r' is appended on a DLO, say $\mathcal{DL}'$. Two records, r and r', are *mutually dependent*, if r depends on r' and r' depends on r. In this section we define a new problem, we term *AtomicAppends*, that captures the properties we need to satisfy when multiple operations attempt to append dependent records on different DLOs.

▶ **Definition 4** (2-AtomicAppends)**.** *Consider two clients, A and B, with mutually dependent records r_A and r_B. We say that records r_A and r_B are* appended atomically *on DLO_A and DLO_B respectively, when:*

- *Either both or none of the records are appended to their respective DLOs (safety)*
- *If neither A nor B fail, then both records are appended eventually (liveness).*

An algorithm *solves* the 2-AtomicAppends problem under a given system model, if it guarantees the safety and liveness properties of Definition 4.

The *k-AtomicAppends* problem, for $k \geq 2$, is a generalization of the 2-AtomicAppends that can be defined in the natural way (k clients, with k records, to be appended to up to k DLOs.) From this point onwards, we will focus on the 2-AtomicAppends problem, and when clear from the context, we will refer to it simply as *AtomicAppends*.

2.4 Communication, Timing and Append Models

The previous subsections are independent of the communication medium, and the failure and timing model. We now specify the communication and timing assumptions considered in the remainder of the paper. We also consider different models on who can append a specific record.

Communication model. We assume a *message-passing* system where messages are neither lost nor corrupted in transit. This applies to both the communication among clients and between clients and DLOs (i.e, the invocation and response messages of the operations).

Timing models. We consider *synchronous* and *asynchronous* systems with respect to both computation and communication. In the former, the evolution of the system is governed by a global clock and a local computation, a message delivery or a DLO operation is guaranteed to complete within a predefined time-frame. For simplicity, we set this time-frame to correspond to one unit of time. In the latter, no timing assumptions are made beyond that they will complete in a finite time.

Append models. We consider three different append models. In the first, and most restrictive one, which we refer to as *Client appends with no delegation*, or ***NoDelegation*** for short, the only way a client can append its record, is by issuing append operations directly to the corresponding DLOs, i.e., no other entity, including the other client, can do so. The second one, referred to as *Client appends with delegation*, or ***WithDelegation*** for short, is a relaxation of the first model, in which one client can append the record of the other client (if it knows it). Finally, in the third model, a record can be appended by an external (w.r.t. the clients) entity, provided it knows the record.

2.5 Client Model and Utility-based Problem Definitions

2.5.1 Client Setting

We assume that clients are ***rational***, i.e., they act selfishly, in a game-theoretic sense, in order to increase their utility [21]. Furthermore, clients are ***risk-averse***, i.e., when uncertain, they prefer to lower the uncertainty, even if this might lower their potential utility [21]; we consider a client to be uncertain when her actions may lead to multiple possible outcomes. To this respect, a rational, risk-averse client runs its own utility-driven protocol that defines its strategy towards a given protocol (game), in such a way that it would not decrease its utility or increase its uncertainty.

Regarding failures, the only type of failure we consider in this work, is ***crash failure***, in which a client might cease operating without any a priori warning.

Under this client model, ***an algorithm $\mathcal{A}$ solves the AtomicAppends problem***, if it provides enough incentive to the clients to follow this algorithm (which guarantees the safety and liveness properties of Definition 4, possibly in the presence of crashes), without any client deviating from its utility-driven protocol. If no such algorithm can be designed, then ***the AtomicAppends problem cannot be solved***.

2.5.2 Utility Models

Looking at the definition of the AtomicAppends problem, one might wonder what is the incentive of the clients to achieve this both-or-none principle on the appends. Let U_X denote the utility function (or incentive) for each client X. A selfish rational client X will try to maximize her utility U_X. Depending on the possible combinations of values the clients' utility functions can take, we can identify a number of different scenarios, we refer as *utility models*. Let us now motivate and specify two such utility models.

Collaborative utility model. Consider two clients A and B that have agreed to acquire a property (e.g., a piece of land) in common, and each has to provide half of the cost. If one of them, say A, pays while B backs off from the deal, then A incurs in expenses while not getting the property. On the other hand, B loses no money in this case, but her reputation may suffer. If both of them back off, they do not have any cost, while if both proceed with the payments then they get the property, which they prefer.

Table 1 The utility of client $X \in \{A, B\}$ in the two utility models considered.

Utility model	Utility of client X
Collaborative	$U_X(\text{both append}) > U_X(\text{none appends}) > U_X(\text{only } \bar{X} \text{ appends}) > U_X(\text{only } X \text{ appends})$
Competitive	$U_X(\text{only } \bar{X} \text{ appends}) > U_X(\text{both append}) > U_X(\text{none appends}) > U_X(\text{only } X \text{ appends})$

If $U_X()$ denotes the utility of agent $X \in \{A, B\}$, then we have the following relations in the scenario described:

$$U_X(\text{both agents pay}) > U_X(\text{no agent pays}) > U_X(\text{only agent } \bar{X} \text{ pays}) > U_X(\text{only agent } X \text{ pays}).$$

In relation to the AtomicAppends problem, record r_A contains the transaction by which client A pays her share of the deal, and the append of r_A in DLO_A carries out this payment. Similarly for client B. So, here we see that under the above utility model, both clients have incentive for both appends to take place. Observe that this situation is similar to the *Coordinated Attack* problem [10], in which two armies need to agree on attacking a common enemy. If both attack, then they win; if only one of them attacks, then that army is destroyed, while the other is disgraced; if none of them attack, then the status quo is preserved.

These utility examples fall in the general utility model depicted in the first row of Table 1, which we call *collaborative*. We will be referring to the AtomicAppends problem under this utility model as the ***Collaborative AtomicAppends*** problem.

Competitive utility model. We now consider a different utility model. Consider two clients A and B that have agreed to exchange their goods. E.g, A gives his car to B, and B gives a specific amount as payment to A. If one of them, say A, gives the car to B, but B does not pay, then A loses the car while not getting any money. On the other hand, B gets the car for free! If both of them back off from the deal, then they do not have any cost. Both proceeding with the exchange is not necessarily their highest preference (unlike in the previous collaborative model).

So, if $U_X()$ denotes the utility of agent $X \in \{A, B\}$, then we have the following relations in the scenario described:

$$U_X(\text{only } \bar{X} \text{ proceeds}) > U_X(\text{both agents proceed}) > U_X(\text{no agent proceeds}) > U_X(\text{only } X \text{ proc.}).$$

In relation to the AtomicAppends problem, record r_A contains the transaction transferring the deed of A's car to B, and the append of r_A in DLO_A carries out this transfer. Similarly, r_B contains the transaction by which client B transfers a specific monetary amount to A (pays for the car), and the append of r_B in DLO_B carries out this monetary transfer. Observe that this scenario is similar to the *Atomic Swaps* problem [13].

These utility examples fall in the general utility model depicted in the second row of Table 1, which we call *competitive*. We will be referring to the AtomicAppends problem under this utility model as the ***Competitive AtomicAppends*** problem.

No matter of the utility, failure or timing model assumed, our objective is to provide a solution to the AtomicAppends problem. Our investigation will focus on identifying the modeling conditions under which this is possible or not, and what is the impact of the model on the solvability of the problem.

3 AtomicAppends in the Absence of Client Crashes

We begin our investigation in a setting with no client crashes, so to study the impact of the utility model on the solvability of the problem.

It is not difficult to observe that in the absence of crash failures, even under asynchrony and NoDelegation, there is a straightforward algorithmic solution to the *Collaborative AtomicAppends* problem: the algorithm simply has client A (resp. client B) issuing operation $append(DLO_A, r_A)$ (resp. $append(DLO_B, r_B)$). Based on Table 1, the clients' utilities are maximized when both append their corresponding records. Since there are no failures and the DLOs are reliable, these operation are guaranteed to complete, nullifying the clients' uncertainty. Hence, the clients will follow the algorithm, without deviating from their utility-driven protocol. This yields the following result:

▶ **Theorem 5.** *Collaborative 2-AtomicAppends can be solved in the absence of failures, even under asynchrony and NoDelegation.*

However, this is *not* the case for the *Competitive AtomicAppends* problem. The problem cannot be solved, even in the absence of failures, in synchrony, and WithDelegation:

▶ **Theorem 6.** *Competitive 2-AtomicAppends cannot be solved in the absence of failures, even in synchrony and WithDelegation.*

Proof. Let us firstly show that client A will never send its record r_A to the other client B. The reason is that this would carry a large risk of B appending r_A itself (and A is risk-averse). Observe that, independently on whether B already appended r_B or not, this would reduce A's utility (see Table 1). Then, we secondly claim that client A will not directly append its own record r_A either. The reason is that, again, independently on whether B already appended r_B or not, this would reduce A's utility (see Table 1). Hence, client A will not have its record r_A appended to DLO_A ever. However, this violates the liveness property of Definition 4, since by assumption neither A nor B fail by crashing. ◀

Note that the above result does not contradict the known solutions for atomic swaps (e.g., [13]), as the primitives used are stronger than the ones offered by DLO (e.g., some form of validation is needed for hashlocks). As we show in Section 5, the problem can be solved in the model we consider, if a reliable external entity is used between the clients and the MDLO. In view of Theorems 5 and 6, in the next section we focus on the study of *Collaborative AtomicAppends* in the presence of crash failures.

4 Crash-prone Collaborative AtomicAppends with Client Appends

In this section we focus on the Collaborative AtomicAppend problem assuming that at least one client may crash, under the NoDelegation and WithDelegation client append models. Observe from Table 1 that both clients have incentive to get both records appended, versus the case of no record appended, with respect to utilities. However, as we will see, in some cases, crashes introduce uncertainty that renders the problem unsolvable.

4.1 Client Appends with No Delegation

We prove that *Collaborative AtomicAppends* cannot be guaranteed by any algorithm $\mathcal{A}$, even in a *synchronous system*, when at least one client crashes and the clients cannot delegate the append of their records.

▶ **Theorem 7.** *When at least one client crashes, Collaborative 2-AtomicAppends cannot be solved in the NoDelegation append model, even in a synchronous system.*

Proof. Consider an algorithm $\mathcal{A}$ that clients can execute without deviating from their utility-driven protocol. Assume algorithm $\mathcal{A}$ solves the Collaborative 2-AtomicAppends problem in the model described. Let E be an execution of algorithm $\mathcal{A}$ in which no client crashes. By liveness, both clients A and B must issue append operations. Consider the first client, say A without loss of generality, that issues the append operation. Let us assume that A issues $append(\text{DLO}_A, r_A)$ at time t. Hence, B issues $append(DLO_B, r_B)$ at time no earlier than t, and A cannot verify that the record r_B is in the corresponding DLO_B until time $t' > t$.

Now consider execution E' of algorithm $\mathcal{A}$ that is identical to E, *up to time* t. Now at time t client B crashes, and hence it never issues $append(DLO_B, r_B)$. Since A cannot differentiate until time t this execution from E, it issues $append(DLO_A, r_A)$ at time t, appending r_A to DLO_A. Even if after time t, A detects the crash of client B, by the specification of NoDelegation, it cannot append record r_B in DLO_B. This, together with the fact that B has crashed, yields that record r_B is never appended to DLO_B, violating safety. Hence, we reach a contradiction, and algorithm $\mathcal{A}$ does not solve the Collaborative 2-AtomicAppends problem. ◀

4.2 Client Appends With Delegation

Let us now consider the more relaxed client append model of WithDelegation. It is not difficult to see that in this model, the impossibility proof of Theorem 7 breaks. In fact, it is easy to design an algorithm that solves the collaborative AtomicAppends problem in a synchronous system, if at most one client crashes. In a nutshell, first both clients exchange their records. When a client has both records, it appends them (one after the other) to the corresponding DLO; otherwise it does not append any record. We refer to this algorithm as Algorithm $\mathcal{A}_{DSync}$ and its pseudocode is given as Code 1. We show:

▶ **Theorem 8.** *In the WithDelegation append model, Algorithm $\mathcal{A}_{DSync}$ solves the Collaborative 2-AtomicAppends problem in a synchronous system, if at most one client crashes.*

Proof. If no client crashes, then the proof of the claim is straightforward. Hence, let us consider the case that one client crashes, say A. There are three cases:

(a) Client A crashes before sending its record. In this case, client B will not append any record and the problem is solved (none case).

(b) Client A crashes after sending its record, but before it does any append. In this case client B will receive A's record and append both records (both case).

(c) Client A crashes after it performs one or two of the appends. Client B will perform both appends, and since DLOs guarantee that a record is appended only once (they are idempotent), the problem is solved (both case).

The above cases and Table 1 suggest that the clients have no risk in running Algorithm $\mathcal{A}_{DSync}$ with respect to their utility-driven protocol. Hence, the claim follows. ◀

We note that algorithm $\mathcal{A}_{DSync}$ solves the problem also in the asynchronous setting, without of course being able to implement the "else" statement (line 5), since in asynchrony, a client cannot distinguish the case on whether the other client has crashed or its message is taking too long to arrive. To this respect, we slightly modify the description of the algorithm to better highlight the inability to detect crashes. We refer to this version of the algorithm as $\mathcal{A}_{DAsync}$; its pseudocode is given as Code 2. We show:

▶ **Theorem 9.** *In the WithDelegation append model, Algorithm $\mathcal{A}_{DAsync}$ solves the Collaborative 2-AtomicAppends problem in an asynchronous system, if at most one client crashes.*

Algorithm 1 $\mathcal{A}_{DSync}$: AtomicAppends WithDelegation, Synchrony, at most one crash; code for Client $X \in \{A, B\}$.

```
1: send r_X to client X̄
2: If r_X̄ is received from client X̄ then
3:     append(DLO_X, r_X)
4:     append(DLO_X̄, r_X̄)
5: Else (client X̄ has crashed)
6:     no append
```

Algorithm 2 $\mathcal{A}_{DAsync}$: AtomicAppends WithDelegation, Asynchrony, at most one crash; code for Client $X \in \{A, B\}$.

```
1: send r_X to client X̄
2: wait until r_X̄ is received from client X̄
3:     append(DLO_X, r_X)
4:     append(DLO_X̄, r_X̄)
```

Proof. As before, we will prove this by case analysis. If no client crashes, then the proof follows easily, given the fact that a DLOs guarantees that a record is appended only once. Hence, let us consider the case that one client crashes, say A. There are three cases:

(a) Client A crashes before sending its record. In this case, client B will not proceed to append any record (none case). Observe that client B might not terminate, but the problem (safety) is not violated.

(b) Client A crashes after sending its record, but before it does any append. In this case client B will receive A's record and append both records (both case).

(c) Client A crashes after it performs one or two of the appends (it means it has sent its record to client B). Client B will perform both appends, and since DLOs guarantee that a record is appended only once, the problem is solved (both case).

The above cases and Table 1 suggest that the clients have no risk in running Algorithm $\mathcal{A}_{DAsync}$ with respect to their utility-driven protocol. Hence, the claim follows. ◀

As already discussed in case (a) of the above proof, it is possible for the client that has not crashed to wait forever, as it cannot distinguish the case when the message is taking too long to arrive and the append operation is taking too long to complete, from the case when the other client has crashed. Hence, algorithm $\mathcal{A}_{DAsync}$, under certain conditions, is *non-terminating*[1].

Furthermore, it is not difficult to see that if both clients fail, neither algorithm $\mathcal{A}_{DAsync}$ nor algorithm $\mathcal{A}_{DSync}$ can solve the Collaborative AtomicAppends problem. For example, in the proof of Theorem 8, in case (b), client B could crash right after appending its own record (i.e., r_B is appended, but r_A is not). This violates safety. In fact, we now show that if both clients can crash, the problem is not solvable, even under synchrony.

▶ **Theorem 10.** *When both clients can crash, the Collaborative 2-AtomicAppends problem cannot be solved WithDelegation, even in a synchronous system.*

Proof. Consider an algorithm $\mathcal{A}$ that clients can execute without deviating from their utility-driven protocol. Assume algorithm $\mathcal{A}$ solves the Collaborative 2-AtomicAppends problem in the model described. Let E be an execution of algorithm $\mathcal{A}$ in which no client crashes. By

[1] Hence, in practice this may force a client to use timeouts in order to avoid blocking forever.

liveness, both records r_A and r_B must be eventually appended. Consider the first record appended, say r_A w.l.o.g., and the client that issued the append operation, say A w.l.o.g.. Let us assume that A issues $append(\text{DLO}_A, r_A)$ at time t. Hence, $append(DLO_B, r_B)$ is issued at time no earlier than t, and A cannot verify that the record r_B is in the corresponding DLO_B until time $t' > t$.

Now consider execution E' of algorithm $\mathcal{A}$ that is identical to E, *up to time* t. Now at time t client B crashes, and hence it never issues $append(DLO_B, r_B)$. Since A cannot differentiate until time t this execution from E, it issues $append(DLO_A, r_A)$ at time t, appending r_A to DLO_A. Then, at time $t+1$ (immediately after $append(DLO_A, r_A)$ completes) A also crashes, and hence never issues $append(DLO_B, r_B)$. Since $append(DLO_B, r_B)$ is never issued, record r_B is never appended to DLO_B, violating safety. Hence, we reach a contradiction, and algorithm $\mathcal{A}$ does not solve the Collaborative 2-AtomicAppends problem. ◀

5 Crash-prone AtomicAppends with SDLO

Theorems 6 and 10 suggest the need to use some external intermediary entity, in order to solve *Competitive AtomicAppends*, even in the absence of crashes, and *Collaborative AtomicAppends*, in the case both clients crash, respectively. This is the subject of this section.

5.1 Smart DLO (SDLO)

We enhance the MDLO with a special DLO, called *Smart DLO* (SDLO), which is used by the clients to delegate the append of their records to the original MDLO. This SDLO is an extension of a DLO that supports a special "atomic appends" record of the form **[client id, {list of involved clients in the atomic append}, record of client]**. When two clients wish to perform an atomic append involving their records and their corresponding DLOs, then they *both* need to append such an atomic appends record in the SDLO; this is like requesting the atomic append service from the SDLO. Once *both* records are appended in the SDLO, then the SDLO appends each record to the corresponding DLO. A pseudocode of this mechanism, together with the client requests, called algorithm $\mathcal{A}_{SDLO}$ is given as Code 3.

Algorithm 3 $\mathcal{A}_{SDLO}$: SDLO mechanism and requests from client $X \in \{A, B\}$; SDLO code only for atomic appends.

```
1: Client X:
2:     append(SDLO, [X, {X, X̄}, r_X])
3:     upon receipt AppendAck from SDLO return
4: SDLO:
5:     Init: S ← ∅
6:     function SDLO.append([X, {X, X̄}, r_X])
7:         S ← S ∥ [X, {X, X̄}, r_X]
8:         if [X̄, {X, X̄}, r_X̄] ∈ S then
9:             append(DLO_X, r_X)
10:            append(DLO_X̄, r_X̄)
11:        return AppendAck
```

So essentially the SDLO.append function in Code 3 can be viewed as a smart contract that "collects" the append requests involved in the AtomicAppends instance and ultimately executes them, by performing individual appends to the corresponding DLOs. Observe that the SDLO does not access the state of DLO_A and DLO_B, but it needs to be able to perform append operations to both of them. In other words, delegation is passed to the SDLO. Also observe that the SDLO returns ack to a client's request, once their atomic appends request is appended in the SDLO, and not when the actual atomic append takes place.

5.2 Solving AtomicAppends with SDLO

It is not difficult to observe that algorithm $\mathcal{A}_{SDLO}$ can solve the AtomicAppends problem in both utility models, even *in asynchrony*, and even if *both clients crash*. Note that $SDLO$, being a distributed ledger by itself, is reliable despite the fact that some servers implementing it may fail (more below). We show:

▶ **Theorem 11.** *Algorithm $\mathcal{A}_{SDLO}$ solves both the Collaborative and Competitive 2-AtomicAppends problems in an asynchronous setting, even if both clients may crash.*

Proof. We consider three cases:

1. If no client crashes, then algorithm $\mathcal{A}_{SDLO}$ trivially solves the problem: Both clients invoke the atomic appends request to the SDLO, these operations complete, and the SDLO eventually triggers the two corresponding appends of records r_A and r_B to DLO_A and DLO_B, respectively (both case).
2. At most one client crashes, say client A. Here we have two cases:
   a. Record $[A, \{A, B\}, r_A]$ is never appended to the SDLO. Since the SDLO will never contain both matching records, it will never append any of the records r_A and r_B (none case).
   b. Record $[A, \{A, B\}, r_A]$ is appended to the SDLO. Since record $[B, \{A, B\}, r_B]$ will eventually be appended by B in the SDLO, it will proceed with the corresponding appends of records r_A and r_B (both case).
3. Both clients crash. If one of the two clients, say A, crashes before appending $[A, \{A, B\}, r_A]$ to the SDLO, then none of the appends of records r_A and r_B will take place in the corresponding DLOs (none case). However, if both clients crash after they have appended the matching atomic appends records, then both records r_A and r_B will be appended by the SDLO (both case).

Observe that the above hold for both utility models. In Competitive AtomicAppends, if a client does not invoke its atomic append request to the SDLO, it knows that the SDLO will not proceed to append the other client's record. This leaves the clients with their second best utility (see Table 1), and hence, both have incentive to invoke the atomic append requests to the SDLO. The reliability of the SDLO nullifies the uncertainty of the clients, and hence they will follow algorithm $\mathcal{A}_{SDLO}$. ◀

Observe that algorithm $\mathcal{A}_{SDLO}$ can easily be extended to solve the *k-AtomicAppend* problem, for any $k \geq 2$, provided that the utility of all records being appended is higher than none being appended for all clients: All clients submit their atomic append request to the SDLO, and then the SDLO performs the corresponding appends. Hence:

▶ **Corollary 12.** *Both the Collaborative and Competitive k-AtomicAppends problems can be solved with the use of SDLO in the asynchronous setting, even if all k clients may crash.*

▶ Remark. As we discussed in the case 2 of the proof of Theorem 11, if client A crashes and record $[A, \{A, B\}, r_A]$ is never appended to the SDLO, none of the records r_A and r_B will be appended. Now, observe that client B can proceed to perform other operations once it has appended $[B, \{A, B\}, r_B]$ (despite the fact that r_B has not been appended to DLO_B, as it is up to the SDLO to do so). Since clients do not need to wait forever for any operation, algorithm $\mathcal{A}_{SDLO}$ is terminating with respect to the clients. Moreover, the SDLO also terminates the processing of all the operations, as long as the appends in other DLOs terminate.

Implementation issues. In the above mechanism and theorem, we treat the SDLO as one entity. Since, however, the SDLO is a distributed ledger implemented by collaborating servers, there are some low-level implementation details that need to be discussed. If we assume that the servers implementing the SDLO are prone to only crash faults and that the SDLO is implemented using an Atomic Broadcast service, as described in [7], then algorithm $\mathcal{A}_{SDLO}$ can be implemented as follows: Clients A and B submit the atomic append requests to all servers implementing the SDLO. Once a server appends an atomic append request record to its local copy of the ledger, it checks if the matching record is already in the ledger. If this is the case, it issues the two corresponding append operations for records r_A and r_B. If up to f servers may crash, then it suffices that $f + 1$ servers, in total, perform these append operations. Given that each record is appended to a DLO at most once (the append operations are idempotent; if a record is already appended, it will not be appended again), it follows that both records are appended in the corresponding DLOs.

6 Conclusion

We have introduced the AtomicAppends problem, where given two (or more in general) clients, each needs to append a record to a corresponding DLO, and do so atomically with respect to each other: either both records are appended or none. We have considered crash-prone, rational and risk-averse clients based on two different utility models, *Collaborative* and *Competitive*, and studied the solvability of the problem under synchrony/asynchrony, different client append models and failure scenarios. Table 2 gives an overview of our results (for two clients): if the problem can be solved, then we list the algorithm we developed, otherwise we use the symbol "✘".

Table 2 Overview of the results. ND stands for NoDelegation and WD for WithDelegation.

<table>
<tr><th colspan="2" rowspan="2"></th><th colspan="3">Synchrony</th><th colspan="3">Asynchrony</th></tr>
<tr><th>ND</th><th>WD</th><th>SDLO</th><th>ND</th><th>WD</th><th>SDLO</th></tr>
<tr><td rowspan="3">Collaborative</td><td>no crashes</td><td>simple</td><td rowspan="2">$\mathcal{A}_{DSync}$</td><td rowspan="5">$\mathcal{A}_{SDLO}$</td><td>simple</td><td rowspan="2">$\mathcal{A}^{(\star)}_{DAsync}$</td><td rowspan="5">$\mathcal{A}_{SDLO}$</td></tr>
<tr><td>up to one</td><td rowspan="2">✘</td><td rowspan="2">✘</td></tr>
<tr><td>both</td></tr>
<tr><td rowspan="3">Competitive</td><td>no crashes</td><td colspan="2" rowspan="3">✘</td><td colspan="2" rowspan="3">✘</td></tr>
<tr><td>up to one</td></tr>
<tr><td>both</td></tr>
<tr><td colspan="8">$^{(\star)}$ might not terminate</td></tr>
</table>

Our results demonstrate a clear separation on the solvability of the problem based on the utility model assumed when appends are done directly by the clients. When appends are done using a special type of a DLO, which we call *Smart* DLO (SDLO), then the problem is solved in both utility models, even in asynchrony and even if both clients may crash.

Our investigation of AtomicAppends did not look into the semantics of the records being appended. Consider, for example, the following scenario. Say that clients A and B initiate an atomic append request with records r_A and r_B, respectively. While the atomic append request is being processed, say by the SDLO, client B appends a record r' directly to DLO_B. It could be the case that the content of record r' is such, that it would affect record r_B. For example, say that the atomic append involves the exchange of a deed of a car with bitcoins; record r_A contains the transfer of the deed and r_B the transfer of bitcoins. If r' involves the withdrawal of bitcoins from the wallet of client B, and this is appended first, then it could

be the case that the wallet no longer contains sufficient bitcoins to carry out the atomic appends request. Even if we enforce the clients to perform all appends – not only atomic appends – through the SDLO (which practically speaking is not desirable), still we need to *validate* records. Therefore, to tackle such cases, we will need to consider *validated* DLOs (VDLOs) [7]. This is a challenging problem, especially in asynchronous settings.

References

1 Atomic swap. https://en.bitcoin.it/wiki/Atomic_cross-chain_trading.
2 Cosmos. https://cosmos.network.
3 Oraclize. http://www.oraclize.it.
4 PolkaDot. https://polkadot.network.
5 Elli Androulaki et al. Channels: Horizontal Scaling and Confidentiality on Permissioned Blockchains. In ESORICS 2018, pages 111–131, 2018. doi:10.1007/978-3-319-99073-6_6.
6 Elli Androulaki et al. Hyperledger fabric: a distributed operating system for permissioned blockchains. In EuroSys 2018, pages 30:1–30:15, 2018. doi:10.1145/3190508.3190538.
7 Antonio Fernandez Anta, Chryssis Georgiou, Kishori Konwar, and Nicolas Nicolaou. Formalizing and Implementing Distributed Ledger Objects. In NETYS 2018, 2018. Also, in SIGACT News, 49(2):58-76, June 2018.
8 Matthew K. Franklin and Gene Tsudik. Secure Group Barter: Multi-party Fair Exchange with Semi-Trusted Neutral Parties. In FC 1998, pages 90–102, 1998. doi:10.1007/BFb0055475.
9 Mimi Gentz and Johnny Dude. Tunable data consistency levels in Microsoft Azure Cosmos DB. https://docs.microsoft.com/en-us/azure/cosmos-db/consistency-levels, June 2017.
10 Piotr J. Gmytrasiewicz and Edmund H. Durfee. Decision-theoretic recursive modeling and the coordinated attack problem. In Proc.of 1992 Conference on AI Planning Systems, pages 88–95. Morgan Kaufmann Publ Inc, 1992.
11 Interledger W3C Community Group. Interledger. https://interledger.org/.
12 Thomas Hardjono, Alexander Lipton, and Alex Pentland. Towards a Design Philosophy for Interoperable Blockchain Systems. *CoRR*, abs/1805.05934, 2018. arXiv:1805.05934.
13 Maurice Herlihy. Atomic Cross-Chain Swaps. In PODC 2018, pages 245–254, 2018. doi:10.1145/3212734.3212736.
14 Maurice Herlihy and Jeannette M. Wing. Linearizability: A Correctness Condition for Concurrent Objects. *ACM Transactions on Programming Languages and Systems*, 12(3):463––492, 1990.
15 Leslie Lamport. How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs. *IEEE Transactions on Computers*, C-28(9):690—691, 1979.
16 N.A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, 1996.
17 Silvio Micali, Michael O. Rabin, and Joe Kilian. Zero-Knowledge Sets. In FOCS 2003, pages 80–91, 2003. doi:10.1109/SFCS.2003.1238183.
18 Andrew Miller, Iddo Bentov, Ranjit Kumaresan, Christopher Cordi, and Patrick McCorry. Sprites and State Channels: Payment Networks that Go Faster than Lightning. *arXiv preprint arXiv:1702.05812*, 2017.
19 Aybek Mukhamedov, Steve Kremer, and Eike Ritter. Analysis of a Multi-party Fair Exchange Protocol and Formal Proof of Correctness in the Strand Space Model. In FC 2005, 2005. doi:10.1007/11507840_23.
20 Arvind Narayanan et al. *Bitcoin and Cryptocurrency Technologies - A Comprehensive Introduction*. Princeton University Press, 2016. URL: http://press.princeton.edu/titles/10908.html.
21 Martin J Osborne et al. *An introduction to game theory*, volume 3. Oxford university press New York, 2004.
22 Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments, 2016. See https://lightning.network/lightning-network-paper.pdf.

23 M. Raynal. *Concurrent Programming: Algorithms, Principles, and Foundations*. Springer, 2013.

24 Michel Raynal. *Distributed Algorithms for Message-Passing Systems*. Springer, 2013. doi:10.1007/978-3-642-38123-2.

25 Ron van der Meyden. On the specification and verification of atomic swap smart contracts. *CoRR*, abs/1811.06099, 2018. arXiv:1811.06099.

26 Matteo Gianpietro Zago. 50+ Examples of How Blockchains are Taking Over the World. *Medium*, 2018. URL: https://medium.com/@matteozago/50-examples-of-how-blockchains-are-taking-over-the-world-4276bf488a4b.

A Smart Contract Oracle for Approximating Real-World, Real Number Values

William George
Kleros Cooperative, Montreal, Canada
william@kleros.io

Clément Lesaege
Kleros Cooperative, Lisbon, Portugal
clement@kleros.io

Abstract

A key challenge of smart contract systems is the fact that many useful contracts require access to information that does not natively live on the blockchain. While miners can verify the value of a hash or the validity of a digital signature, they cannot determine who won an election, whether there is a flood in Paris, or even what is the price of ether in US dollars, even though this information might be necessary to execute prediction market, insurance, or financial contracts respectively.

A number of promising projects and research developments have provided a better understanding of how one might construct a decentralized, binary oracle - namely an oracle that can respond by one of two possibilities, typically "yes" or "no", even while not requiring the interaction of a trusted third party. In this work, we extend these ideas to construct a general-purpose, decentralized oracle that can estimate the value of a real-world quantity that is in a dense totally ordered set, such as $\mathbb{R}$. In particular, this proposal can be used to estimate real number valued quantities, such as required for a price oracle. We will establish a number of desirable properties about this proposal. Particularly, we will see that the precision of the output is tunable to users' needs.

2012 ACM Subject Classification Theory of computation → Algorithmic game theory and mechanism design; Security and privacy → Distributed systems security

Keywords and phrases price oracle, Ethereum, blockchain

Digital Object Identifier 10.4230/OASIcs.Tokenomics.2019.6

1 Introduction and related work

Blockchains and, specifically, smart contract platforms such as Ethereum [2], provide significant opportunities for systems that transfer value in a trustless way. However, the inability of blockchains to natively observe events in the outside world has limited this potential. While miners can verify computations, such as required for validating a digital signature, as part of the protocol they perform, this protocol generally does not have access to off-chain information, such as weather data or even prices of blockchain-based assets in USD terms that may be required for contracts such as flood insurance or financial contracts to perform properly. Indeed, while new models of economic relationships have been seen to be facilitated by smart contracts [20, 11], such smart contracts will likely often require access to off-chain information. Already in the Ethereum white paper [2], the need for oracles, i.e. mechanisms that can import external information on-chain, is discussed as necessary to overcome the limitation of "value-blindness" of financial contracts towards crypto-assets. The limitations on the types of applications that are possible on smart contract platforms that are imposed by the difficulty in obtaining adequate oracles have since remained an active source of discussion in the blockchain community [9, 13].

Town Crier [21] and TLS Notary [6] allow for demonstrating on-chain that some information has been published on a given TLS enabled website. However, this approach requires that the website be trusted to honestly provide the required information; hence it is not appropriate for all use cases. Furthermore, this style of oracle can be vulnerable to insider attacks where a malicious employee of whatever entity controls the website can influence the information that is used on-chain. ChainLink [14], which has acquired Town Crier [12], proposes a model in which data drawn from multiple third party sources can be aggregated and a reputation system is used to evaluate different sources.

A contrasting approach is taken by decentralized oracles, namely oracles that do not depend on any trusted third party with special privileges. These systems typically involve setting up some economic game. Then, the incentive structure of this game is designed so that if participants follow their incentives, the oracle will produce correct answers.

The most successful example of a price oracle to date is likely that which is used by Maker DAO [16] so that its stablecoin Dai can remain pegged to the US dollar. As Maker DAO is based on a system of collateralization, and collateral is in ether, an oracle that can give the price of ether in USD is required. Maker DAO uses a median of values provided by trusted authorities such as leading exchanges, which are chosen (and can be replaced) by MKR token holders. Thus, this oracle is decentralized in the sense that it is ultimately responsive to token holders, albeit via a delegated system. It is worthwhile to note that, in this case, MKR holders have a strong incentive to choose good oracles as the usefulness of Dai as a stablecoin drives demand for MKR tokens, which must be burned to pay a fee when recovering collateral. While this system has contributed to the impressive stability of Dai, even in the recent cryptocurrency bear market, it is not clear how to generalize this idea to price oracles not built around a widely-used stablecoin.

Of particular note are the existing oracles based on the concept of Schelling points [18]. Here, users submit answers for what the output of the oracle should be and are rewarded if they are coherent with the majority and/or punished if they are incoherent. An early proposal for a Schelling point based oracle was Truthcoin [19]. The underlying idea, is then that users will submit true answers because they expect other users to also submit true answers. An attempt to apply this idea to price oracles discussed in [10] is to have users submit a value and then outputting the median submission. Then submitters are penalized if their submission falls outside of the 25th to 75th percentile range of all submissions, ideally encouraging coherence around the true result. As already discussed in [10], this first version of a Schelling point based price oracle is potentially vulnerable to "micro-cheating" as submitters risk little penalty if they provide small variations on the true value, as long as this variation is not too extreme. Such "micro-cheating" may allow an attack to nudge the output of the oracle and potentially affect the decision making of other submitters, leading to more substantially deviations over time.

A notable example of a system that builds upon the ideas of Truthcoin is Augur [17]. Augur provides oracles for prediction markets by allowing holders of the Augur token REP to dispute responses provided by default responders designated by the creator of the market. Augur allows question askers to create markets where the answer can be binary, multiple choice, or scalar (i.e. drawn from $\mathbb{R}$). Notably, when creating a scalar prediction market, the asker must currently specify a minimum and maximum value, as well as a precision, which somewhat limits the flexibility of such markets. Moreover, note that Augur itself requires knowledge of the price of the Augur token REP in order to determine if its fees should be adjusted up or down in order to perform a "market cap nudge" so that buying a majority of the REP to perform a 51% attack is financially not worthwhile relative to the amount of

value at stake in Augur prediction markets. This would be a natural use case for Augur's own scalar oracle. However, for the moment, Augur has opted to use a delegated system based on a collection of trusted third parties to determine the price of REP for these fee adjustments and considers eventually using its scalar oracle for this purpose an "active area of research" [5, 17].

Similar to Augur, Gnosis [4] offers a prediction market system based on oracles. Gnosis is "oracle agnostic," meaning that a Gnosis prediction market contract can reference any oracle [8]. That said, the Gnosis team has developed oracles themselves which can be used for their markets, such as their "ultimate oracle" mechanism [3]. Gnosis allows for prediction markets based on scalar values such as prices, by proposing ultimately binary questions such as whether the price will rise or fall [8].

Kleros [15] is a dispute resolution system on Ethereum based on the idea that parties can enter into business relationships for which any financial transfers are held in escrow in an Ethereum contract. Then, in case of dispute between the parties, a number of "jurors" are randomly drawn via a token weighted system to rule on the case and are incentivized via a Schelling point based mechanism. Already, such a system is a kind of oracle as it brings knowledge of the honest party in these cases on-chain, and the Kleros white paper [15] envisages asking jurors a wide variety of questions, so that Kleros can act as a general oracle. At its current stage of development [1], Kleros allows its jurors to rule between a finite set of predetermined options, particularly allowing binary choices. A notable feature of Kleros, as the cases that are considered by Kleros may require a substantial per-juror effort to be analyzed, is that it includes an appeal system so that juror effort is minimized even as the security against potential attacks should scale with the number of jurors that would be involved in a potential appeal.

Finally, ASTRAEA [7] proposes an oracle capable of providing a binary outputs and provides a rigorous analysis of the security properties of their system. ASTRAEA makes use of two groups – "voters" and "certifiers" – with different incentive structures that must agree for the oracle to return a value. This departs somewhat from the structure of the Schelling point based systems described above, but uses similar ideas.

2 Our contribution

We assume the existence of a decentralized, general purpose oracle that is capable of deciding between binary propositions, namely it is capable of producing true answers to statements about the external world such that the response is either "yes" or "no." For example, one could use Kleros [15], Augur [17], or ASTRAEA [7] to play this role, inheriting their respective security models and guarantees. Then we propose a way to extend such a binary oracle into an oracle capable of producing an element in a dense totally ordered set - namely a totally ordered set such that if x and y are in the set, there exists some z in the set such that $x < z < y$. In particular, as the set of real numbers $\mathbb{R}$ is an example of such a set, our oracle can be used to determine the price of some asset in a way that can be used on-chain. This oracle remains decentralized and general purpose. We establish a number of desirable properties about this oracle, particularly that the ultimate precision of the output dynamically adjusts to the greatest level of precision that is demanded by a user that is ultimately determined to be honest. Hence, honest users can force very precise outcomes (see Theorem 11) while limiting the ability of hostile users to delay the functioning of the oracle or otherwise consume network resources (see Propositions 8 and 9).

3 Proposal

We take $\mathcal{S}$ to be a dense totally ordered set (such as $\mathbb{R}$).

3.1 Assumptions on the underlying binary oracle

We suppose we have access to a pre-existing oracle that can decide binary propositions about the real-world. Namely, suppose we have two statements $\mathcal{A}$ and $\mathcal{B}$, one of which corresponds to reality. Then the binary oracle:

$$\mathcal{O}_B : \{\text{possible submissions to a (cryptoeconomic) game}\} \to \{\mathcal{A}, \mathcal{B}\}$$

will return one of $\mathcal{A}$ or $\mathcal{B}$ in a way that can be computed on-chain, after a delay of t Ethereum blocks for the (cryptoeconomic) game to be played, and in exchange for paying some fee A.

Furthermore, we allow for the possibility that the binary oracle has an appeal mechanism, which may cause an additional time delay and additional fees. Namely, we suppose that any actor can appeal a ruling of the binary oracle in exchange for paying an additional fee, $f_{\mathcal{A},i}$ or $f_{\mathcal{B},i}$, when one is in the ith appeal round and is staking on the claim that $\mathcal{A}$ or $\mathcal{B}$ is true respectively. One might want to have $f_{\mathcal{A},i} \neq f_{\mathcal{B},i}$, for example, to require higher appeal fees for the side that lost the previous round. If this fee is paid on behalf of one "side," i.e. is staked on the truth of $\mathcal{A}$ or $\mathcal{B}$, the corresponding fee must be paid for the other side as well (with the potential for multiple actors to pay this fee collectively). If one side pays an appeal fee and not the other, the side that pays its fees is considered to be the result of the binary oracle. If both sides pay their fees, the binary oracle rules again (with the idea that more resources can be put towards this ruling, so ideally it is more likely to be correct). Whatever fees that are paid on behalf of the side that is the ultimate output of the oracle after all appeals are refunded, whereas fees paid on behalf of the other side are lost (with the potential that they are at least partially redistributed to the fee payers of the winning side).

Appeals result in delaying the result of the oracle by an additional t Ethereum blocks per appeal, up to some maximum number of appeal rounds. In Section 5 we will assume bounds on the growth of appeal fees and consider resulting bounds on the attacker's ability to delay the result of the oracle in terms of her resources.

This proposal was originally developed as an extension to Kleros [15]. As a result, our assumptions above on the structure of eventual appeals are modeled on the Kleros system. The Kleros fee model [1] is designed so that there is an incentive to pay fees on behalf of outcomes that one thinks are likely to ultimately to be chosen by the oracle because one can win some of the fees staked by the losing side. Indeed, Kleros envisages the participation of fee insurers whose economic model is to pay fees on behalf of cases they deem worthy, reducing the practical inconvenience of requiring both sides of the case to pay fees.

However, the successive dispute rounds used in Augur [17] can also be thought of as appeals satisfying this structure (taking two consecutive dispute rounds together and thinking of them as a single appeal), with their "forking market" representing a decision that has reached the maximum number of appeals. Moreover, our results should be adaptable to binary oracles with differing appeal systems. Indeed, one can recover the situation of a binary oracle without an appeal mechanism by just considering that the maximum number of appeals is zero. Hence our results apply equally to such systems.

3.2 Discussion of actors and attack model

The principal actors of our proposal, beyond whatever actors participate in the underlying binary oracle that is used, are respondents, who submit information about $v \in \mathcal{S}$, the value that the oracle is attempting to determine. Specifically each respondent submits an interval in

which they believe that v belongs. Respondents pay a deposit D which they risk losing if the information provided is ultimately judged to be incorrect, see Section 4. While respondents may obtain a reward if the information they provide is judged to be correct, we will see that they are not on average compensated by our system. Hence, we expect that the primary motivation of respondents is some external interest in the result the oracle produces. We will see, under some idealized assumptions about the performance of the $\mathcal{O}_B$, that it is sufficient for there to be a single honest respondent for our oracle to produce honest results, see Theorem 11. For an oracle that is part of a decentralized application with wide use, participation of respondents in these conditions is not an unrealistic assumption. However, an investigation of additional ways to incentivize the participation of respondents may be a subject for future work.

In this work, we will consider attacks from attackers that have the capacities of respondents. Namely, we will analyze attacks that submit malicious responses or call appeals in a hostile manner. Of course, the quality of the results of our real-valued oracle depends on the quality of the results of the underlying binary oracle. As we allow for the possibility of using any binary oracle, we do not directly consider an attack model where it is possible to corrupt the results of the binary oracle. However, in our results, it will be clearly indicated when one must make hypotheses about the accuracy of $\mathcal{O}_B$. Furthermore, we do not consider attacks on the underlying infrastructure of the smart contract platform, such as 51% attacks or network denial-of-service attacks on Ethereum.

3.3 Proposed oracle algorithm

The procedure we propose to approximate the true value of some quantity is based on a sort of modified binary search of the responses, where, rather than split the list of responses at the median when performing a comparison, we detect incoherences that prevent a consensus answer from being accepted and then take a comparison with respect to the median of the list of these incoherences. We are performing these operations on elements in $\mathcal{S}$, which a priori is not closed under averaging, so the normal median may not be defined. However, if we need to take the median of a set D with an even number of elements, namely in the case that requires computing an average, as $\mathcal{S}$ is a dense totally ordered set, we can find some (not necessarily unique) element z of $\mathcal{S}$ such that half of the elements of D are on either side of z. We suppose that for a given $\mathcal{S}$ one has some way of efficiently choosing such a z, and we consider it to be a median of D. In the remainder of this paper, in the context of generic dense totally ordered sets, we use the words median and average in this sense. In the case $\mathcal{S} = \mathbb{R}$, we take the normal median.

In detail, we consider the following:

▶ **Algorithm 1.** *Input: Each respondent $\mathcal{USR}_i$ submits two distinct values - a lower bound $l_i \in \mathcal{S}$ and an upper bound $u_i \in \mathcal{S}$, $l_i < u_i$, giving an interval (l_i, u_i) in which this respondent believes the true value of the question is located.*

- *Sort the lower bound responses into a list $\mathcal{L}$ and the upper bound responses into a list $\mathcal{U}$, where in each case identical values are considered as single elements.*
- *Compute the lists*

$$\mathcal{L}_0 = \{l_i \in \mathcal{L} : \exists u_j \in \mathcal{U},\ u_j \leq l_i,\ \nexists\ l_k \in [u_j, l_i) \cap \mathcal{L}\}$$

and

$$\mathcal{U}_0 = \{u_i \in \mathcal{U} : \exists l_j \in \mathcal{L},\ l_j \geq u_i,\ \nexists\ u_k \in (u_i, l_j] \cap \mathcal{U}\}\,.$$

- *Compute*

  $$\mathcal{C}_0 = \{median(l_i, u_j) : l_i \in \mathcal{L}_0,\ u_j \in \mathcal{U}_0,\ u_j \leq l_i,\ \nexists\ l_k \in [u_j, l_i) \cap \mathcal{L},\ \nexists\ u_k \in (u_j, l_i] \cap \mathcal{U}\}$$

  (So, if we considered $\mathcal{L}$ and $\mathcal{U}$ in the same line, essentially $\mathcal{L}_0$ would consist of lower bounds which have an upper bound to their immediate left and $\mathcal{U}_0$ would consist of upper bounds that have a lower bound to their immediate right. Then $\mathcal{C}_0$ consists of the midpoints between each of these pairs.)
- *If $\mathcal{C}_0 \neq \emptyset$*
  - *For each $z \in \mathcal{C}_0$ perform the following in parallel:*
    - *Ask the binary oracle $\mathcal{O}_B$ if*

      desired value $\leq z$

      or

      desired value $> z$.

    - *Allow appeals of their decision as necessary following the fee structure described in Section 3.1, where here the two sides are as follows:*

      desired value $\leq z$

      or

      desired value $> z$

      - *If one side pays its required fees but not the other, $\mathcal{O}_B$ is considered to rule in favor of the side that paid its fees.*
      - *If neither side pays its fees, the previous ruling stands.*
  - *Take $\mathcal{C}_1 = \mathcal{C}_0$.*
  - *While $\mathcal{C}_1 \neq \emptyset$*
    - *If $\#\mathcal{C}_1$ is odd, calculate $m = median(\mathcal{C}_1) \in \mathcal{C}_1$. If $\#\mathcal{C}_1$ is even, choose one of the two middle-most values of $\mathcal{C}_1$ as m in some predictable way (such as by always taking the value on the left).*
    - *Eliminate all l_i and u_i that are on the wrong side of what $\mathcal{O}_B$ decided with respect to m (taking into account the final outcome after any appeals) from $\mathcal{L}$ and $\mathcal{U}$.*
    - *Add m to $\mathcal{L}$ if $\mathcal{O}_B$ has ruled that the true value is higher than m, and add m to $\mathcal{U}$ otherwise.*
    - *Recalculate $\mathcal{L}_0$ and $\mathcal{U}_0$ based on the updated $\mathcal{L}$ and $\mathcal{U}$.*
    - *(Re)calculate $\mathcal{C}_1$ as*

      $$\{median(l_i, u_j) : l_i \in \mathcal{L}_0,\ u_j \in \mathcal{U}_0,\ u_j \leq l_i,\ \nexists\ l_k \in [u_j, l_i) \cap \mathcal{L},\ \nexists\ u_k \in (u_j, l_i] \cap \mathcal{U}\}.$$

Output the average of the largest remaining element of $\mathcal{L}$ and the smallest remaining element of $\mathcal{U}$. Payments are made to respondents according to a structure that will be described in Section 4.

The respondent $\mathcal{USR}_i$ is ruled incorrect and loses his deposit if the final output of Algorithm 1, $v_{\text{output}} \notin (l_i, u_i)$. See Section 4 for details on the payment made to a respondent $\mathcal{USR}_i$ for whom $v_{\text{output}} \in (l_i, u_i)$.

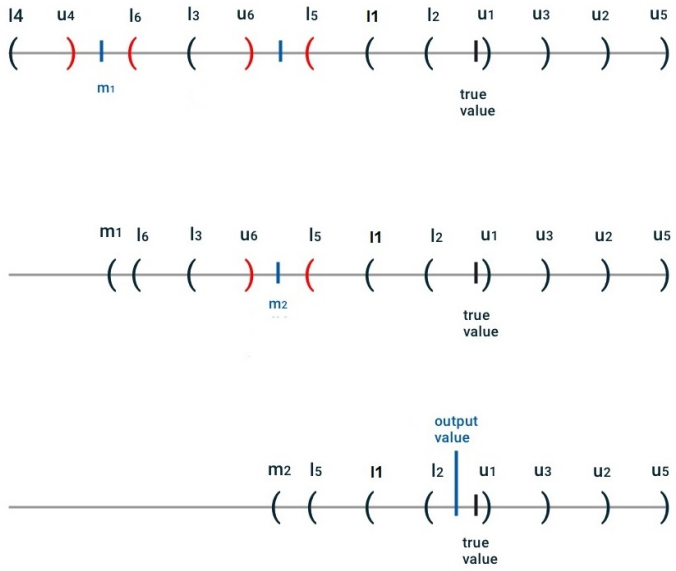


Figure 1 An example of Algorithm 1. Six respondents $\mathcal{USR}_i$ each submit (l_i, u_i). The elements of $\mathcal{C}_0$ are marked by the short, straight blue lines. The binary oracle $\mathcal{O}_B$ is called to rule on each element of $\mathcal{C}_0$ in parallel, and then these results are translated into the output via two rounds of the while loop. The three lines show the state of $\mathcal{L} \cup \mathcal{U}$ before the first round, between the two rounds, and after the second round respectively. The execution shown corresponds to $\mathcal{O}_B$ ruling that the true value v is such that $v > m_1$ in round one and $v > m_2$ in round two.

We will choose the respondent deposits to be large enough so that the deposits of respondents who are ruled incorrect is sufficient to cover the fees required for the initial ruling (i.e. excluding potential appeals) of each call of $\mathcal{O}_B$ required by the for loop, namely at every point in $\mathcal{C}_0$. This will require that the deposit D from each respondent be greater than or equal to A, the total amount of fees required by a single, non-appealed call of $\mathcal{O}_B$. In contrast, the fees for any appeals are submitted independently from the respondent deposits (depending on the structure of the cryptoeconomic game used by $\mathcal{O}_B$, this will typically be done by parties interested in winning the stake of the other side as discussed in Section 3.1).

▶ **Proposition 1.** *Enough fees are paid by respondents who are ultimately ruled incorrect to cover the initial round of all required calls to the binary oracle. Specifically,*

$$\#\text{ submissions ruled incoherent} \geq \#\mathcal{C}_0 = \#\text{ rulings required}$$

Proof. As each respondent pays a deposit that includes A, the cost of a call to $\mathcal{O}_B$ before appeals, there is $A \cdot (\#$ submissions ruled incoherent$)$ available to cover the fees of the total initial round of binary oracle calls. By construction, there is a ruling with respect to each point in $\mathcal{C}_0$. So it suffices to prove the first inequality.

Take $c \in \mathcal{C}_0$ such that $c \geq v$, where v is the ultimate output of the oracle. For each such c there are some $l_i \in \mathcal{L}_0$, $u_j \in \mathcal{U}_0$ such that $u_j \leq c \leq l_i$ and there is no $l_k \in (u_j, l_i) \cap \mathcal{L}$. Then l_i was the lower bound of an incoherent submission.

We claim that this process produces a distinct l_i for each $c \in \mathcal{C}_0$. Indeed, if c_1, $c_2 \in \mathcal{C}_0$ are as above with $u_{j,1} \leq c_1 \leq l_{i,1}$ and $u_{j,2} \leq c_2 \leq l_{i,2}$ but $l_{i,1} = l_{i,2}$, then either

- $u_{j,1} \in (u_{j,2}, l_{i,2}]$ which contradicts the definition of $\mathcal{C}_0$
- $u_{j,2} \in (u_{j,1}, l_{i,1}]$ which again contradicts the definition of $\mathcal{C}_0$ or
- $u_{j,1} = u_{j,2}$ which, as $l_{i,1} = l_{i,2}$, implies that $c_1 = c_2$.

Similarly, for each element of $\mathcal{C}_0$ less than v there corresponds some $u_j \in \mathcal{U}_0$ that is the upper bound of an incoherent submission. ◀

We see that all of the rulings of the binary oracle that are needed to evaluate the while loop were, in fact, decided.

▶ **Proposition 2.** *During the while loop, each $\mathcal{C}_1$ that is computed is a subset of $\mathcal{C}_0$. Hence, for each m computed during the loop, $\mathcal{O}_B$ will have ruled either*

- *desired value $\leq m$ or*
- *desired value $> m$*

Proof. The sets $\mathcal{L}$ and $\mathcal{U}$ are only modified during the while loop. There, if $\mathcal{O}_B$ rules that desired value $\leq m$, all elements greater than or equal to m are eliminated from $\mathcal{L}$ and $\mathcal{U}$, and m is added to $\mathcal{U}$. Due to the local way that $\mathcal{C}_1$ is defined, as the elements of $\mathcal{L}$ and $\mathcal{U}$ less than m remain unchanged, the only way a new element could be added to $\mathcal{C}_1$ is if there existed some $l_i \in \mathcal{L}_0$, $m \leq l_i$ such that median$(l_i, m) \in \mathcal{C}_1$. However, when computing $\mathcal{C}_1$, m is a strict upper bound for $\mathcal{L}$, so there will not exist any such l_i. A similar argument applies if $\mathcal{O}_B$ had ruled that desired value $> m$. ◀

▶ Remark 3. Note that is possible that $\mathcal{O}_B$'s rulings on different points in $\mathcal{C}_0$ will be incoherent; e.g. one call of $\mathcal{O}_B$ will rule that $v \leq m_1$ and another call of $\mathcal{O}_B$ will rule that $v > m_2$ even if $m_2 \geq m_1$. This does not prevent the algorithm from halting as the while loop gives priority to the decisions required along the path of a binary search.

▶ Remark 4. At the expense of additional gas, after each appeal round in algorithm 1, one can test whether the required $\mathcal{O}_B$ calls for the while loop to terminate have been finalized, i.e. have not been appealed. Depending on how underlying binary oracle it structured, it may be necessary to resolve all of the appealed calls of $\mathcal{O}_B$ for an appropriate payment of its internal incentives, however this need not unnecessarily delay the finalization of the result of Algorithm 1.

A priori, it is conceivable that at some point of Algorithm 1, $\mathcal{L}$ and $\mathcal{U}$ become empty and the last step of the algorithm fails. We see that this cannot, in fact, occur.

▶ **Proposition 5.** *Suppose that there is at least one submission (l_*, u_*) to the oracle. Then if Algorithm 1 halts, it returns a value in $\mathcal{S}$.*

Proof. We will see that neither $\mathcal{L}$ nor $\mathcal{U}$ becomes the empty set. Then, in particular, the last step of Algorithm 1 is well-defined because there *is* a largest remaining element in $\mathcal{L}$ and a smallest remaining element of $\mathcal{U}$.

We argue inductively that $l = \min\{\mathcal{L} \cup \mathcal{U}\}$ is in $\mathcal{L}$ after each round of the while loop. Before the first round, this is clearly true, as we have assumed there is at least one submission, and for each submitted element of $\mathcal{U}$ there is a corresponding, smaller element of $\mathcal{L}$. In each round of the while loop, after the appropriate value of m is calculated, either

- $\mathcal{O}_B$ rules that desired value $> m$ or
- $\mathcal{O}_B$ rules that desired value $\leq m$.

In the first case, m is added to $\mathcal{L}$, and it also becomes $\min\{\mathcal{L} \cup \mathcal{U}\}$. In the second case, $\min\{\mathcal{L} \cup \mathcal{U}\}$ is left unchanged, and hence in $\mathcal{L}$. Hence $\mathcal{L}$ never becomes the empty set. A similar argument shows $\mathcal{U} \neq \emptyset$. ◀

Moreover, thinking of $\mathcal{C}_1$ as a set of inconsistencies that prevents the oracle from finding an answer that is a consensus among the different responses, we see that the number of these inconsistencies is reduced by half after each round of the while loop.

▶ **Lemma 6.** *Each round of the while loop reduces the length of $\mathcal{C}_1$ by at least half.*

Proof. Consider a given round of the while loop. Denote $n = \#\mathcal{C}_1$ in this round. Note that the addition of m to $\mathcal{L}$ or $\mathcal{U}$ after the call to $\mathcal{O}_B$ cannot create any new elements of $\mathcal{C}_1$, as m becomes either the smallest element of $\mathcal{L}$ or the largest element of $\mathcal{U}$.

If n is odd, then $\frac{n-1}{2}$ elements in $\mathcal{C}_1$ are on either side of m. Hence at least this many elements are eliminated, whichever way $\mathcal{O}_B$ rules. Moreover, m itself is given as the median of an upper bound u_i to the left and a lower bound l_j to the right. Depending on the ruling of $\mathcal{O}_B$, either u_i or l_j is eliminated. So m is also removed from $\mathcal{C}_1$.

If $n = 2k$ is even, then one of the two most central elements of $\mathcal{C}_1$ is used as m. Thus, on one side of m there will be $k-1$ elements of $\mathcal{C}_1$ and on the other k. Hence there are at least k elements that are eliminated by the choice of $\mathcal{O}_B$, including m itself which again is also removed from $\mathcal{C}_1$. ◀

Lemma 6 has the immediate consequence that:

▶ **Corollary 7.** *Algorithm 1 halts.*

In Section 6, we will consider more precise estimates on the complexity of Algorithm 1.

4 Incentivizing respondents to submit short intervals

In this section we will describe what payouts are made to respondents based on the results of Algorithm 1. For the moment, as discussed in Section 3.2, we imagine that there is no up-front cost paid by the party that sets up whatever application that requires the oracle of Algorithm 1. Indeed, the cost of any initial round (excluding appeals) calls to $\mathcal{O}_B$ that are required in the execution of Algorithm 1 is paid by the respondents. While appeal costs may be covered by other parties (such as the fee insurers imagined by Kleros [1]), one would expect that most appeal fees would also be covered by respondents. Then, while some respondents may make gains from paying fees for the position that is ultimately coherent and winning stake from respondents who lose their deposits, the amount that must be paid to $\mathcal{O}_B$ will collectively mostly come from the respondents. Thus, in terms of the internal incentives of Algorithm 1, the respondents are playing a negative sum game, and in particular are not, on average, compensated for their efforts. In the event that there are many parties with an interest in the result of the oracle, it is nonetheless not unreasonable to expect submissions from respondents in this setting. Effective models for having an "Asker" that pays a fee which can cover compensation for respondents is a potential subject for future work.

Recall, a respondent places a deposit D which he loses if the interval he submits (l_i, u_i) does not contain the output of Algorithm 1. A priori a user might then want to submit a very large interval such as $(-\infty, \infty)$ in order to be guaranteed to be correct. (Note again, parties with an external financial interest in the result of the oracle may nonetheless want an incentive to submit useful estimates). We will design the redistribution mechanism so that respondents have an incentive to submit more precise estimates. To do this we will weight their payouts by an inverse exponential of the length of the submitted intervals.

Consider a respondent $\mathcal{USR}_i$ who submits an interval $I_i = (l_i, u_i)$. If $\mathcal{S}$ is a metric space, such as $\mathbb{R}$, one can take $\text{length}(I_i) = u_i - l_i$ (or more generally as the distance from l_i to u_i). Then, if the ultimate response to the oracle is not in I_i, the user loses his deposit D. If the response is in I_i, the user receives

$$\frac{\#\text{ incorrect responses} \cdot D - \text{cost of first round } \mathcal{O}_B \text{ fees}}{\sum_{j \text{ such that } \mathcal{USR}_j \text{ correct}} \alpha^{-\text{length}(I_j)}} \cdot \alpha^{-\text{length}(I_i)}, \tag{1}$$

where $\alpha > 1$ is some fixed constant. Note that this quantity is positive by Proposition 1. If $\mathcal{S}$ is not a metric space, then the payoff can be split equally between correct respondents.

As such, the sum of the lost deposits from the incorrect users is equal to the sum of the payouts to the correct respondents plus the amount required to pay the fees required by $\mathcal{O}_B$ in the first round of each call. Notice that if a respondent submits an infinite length interval such as $I_i = (-\infty, \infty)$, then $\alpha^{-\text{length}(I_i)} = 0$, for payoffs given by formula 1. So the respondent obtains no reward but suffers no penalty. On the hand, respondents who submit more precise intervals obtain higher rewards.

In Section 8 we will analyze respondent behavior under the incentives given by formula 1. We will show that, under hypotheses on α and certain heuristic assumptions about the behavior of $\mathcal{O}_B$, that it is also not profitable for respondents to submit intervals that are smaller than the precision with which a respondent could reasonably know the desired value. In general, it would be an interesting subject for future experiments to determine how tweaking the weighting of these payoffs, particularly the constant α, would change user behavior to encourage them to submit precise answers for high payoffs, accepting the risk that a very precise answer risks being ruled incorrect even if answered in good faith.

5 Bounds on time griefing in terms of attacker resources

In this section, we estimate an attacker's ability to delay the oracle as a function of her resources. First note,

▶ **Proposition 8.** *If all intervals submitted by the respondents are correct, that is to say the true value $v \in I_i$ for all i, then no calls to $\mathcal{O}_B$ are required.*

Proof. If $v \in I_i = (l_i, u_i)$ for all i, then $l_i < v$ for all v. Similarly $v < u_j$ for all j. Hence $\#\mathcal{C}_0 = \emptyset$ and no calls to $\mathcal{O}_B$ are made. ◀

Now we consider situations where we have an attacker. All of the calls of $\mathcal{O}_B$ of the for loop are performed in parallel, however an attacker can attempt to delay the result of the oracle by appealing one or more of these decisions. As discussed in Section 3.1, each appeal round of $\mathcal{O}_B$ takes t time and all other operations take negligible time. Again, we take that the fees for an initial round ruling of $\mathcal{O}_B$ to be A.

Suppose that appeal fees are such that

$$\min\left\{\sum_{j=1}^{i} f_{\mathcal{A},j}, \sum_{j=1}^{i} f_{\mathcal{B},j}\right\} \geq K \cdot A \cdot 2^i,$$

for all i, where $K > 0$ is a constant that depends only on what algorithm is used for the underlying binary oracle. This is particularly the case if one uses either Kleros [15] or Augur [17] as the binary oracle.[1] Denote by R the attacker's financial resources.

▶ **Proposition 9.** *Suppose that all responses submitted other than the attacker's are ruled correct, namely the output value is in the submitted interval, and suppose that $f_{\mathcal{A},i}$ and $f_{\mathcal{B},i}$ are as above. Then, the maximum number of appeals required is $O_A(\log_2(R))$, and hence the maximum amount of time an attacker can delay a result is $O_A(t \cdot \log_2(R))$, where the implicit constants are allowed to depend on A.*

[1] This is explicitly the case for Kleros [1]. In Augur [17], applied to a binary decision between $\mathcal{A}$ and $\mathcal{B}$, the dispute bond required to dispute a pending outcome of $\mathcal{B}$ is $2S(\mathcal{B}, n) - S(\mathcal{A}, n) \geq S(\mathcal{B}, n) \geq S(\mathcal{A}, n)$, where $S(*, n)$ denotes the total amount of REP staked on choice $*$ prior to the nth round. Hence, $f_{\mathcal{A},i} \geq \sum_{j=1}^{i-1} f_{\mathcal{A},j}$ $\forall i$. Then the bound follows from a standard induction argument.

Proof. If the attacker repeatedly appeals a given decision, then the appeal fees through the mth appeal are at least $K \cdot A \cdot 2^m$. Then, even if the attacker uses all of her resources in appealing a single decision, we have

$$K \cdot A \cdot 2^m \leq \text{money spent by Attacker} \leq R \Rightarrow m \leq \log_2\left(\frac{R}{K \cdot A}\right) = O_A(\log_2 R).$$

(Note that as $\mathcal{O}_B$ is assumed to be always correct, the attacker will ultimately lose her appeal so she these resources will, in fact, be consumed.) ◀

6 Running time and number of calls to binary oracle

In Section 5, we examined an attacker's ability to delay the execution of the oracle by forcing additional appeals. In this section, we will examine the effect of one or more incoherent respondents on the running times of the for and while loops. This is particularly relevant in evaluating the gas costs of Algorithm 1.

Recall, the cost of submitting an incorrect response (l_{attack}, u_{attack}) is D, via a lost deposit when it is eventually determined that the ultimate answer to the oracle is not in the submitted interval. Once again, we denote by R the collective financial resources of respondents who submit intervals that are ultimately ruled to be incoherent, which without loss of generality we can assume to all be controlled by a single attacker.

▶ **Proposition 10.** *Suppose that all responses submitted other than the attackers are ruled correct, namely the output value is in each of these intervals. Then there are at most* $\frac{R}{D}$ *many calls to* $\mathcal{O}_B$ *that must be resolved during the for loop. Consequently, the while loop requires at most* $\max\{\log_2(\#\mathcal{C}_0) + 1, 0\} \leq \max\left\{\log_2\left(\frac{R}{D}\right) + 1, 0\right\}$ *rounds.*

Proof. The attacker can only place at most $\frac{R}{D}$ incorrect solutions. So, by Proposition 1,

$$\#\text{ rulings required} = \#\mathcal{C}_0 \leq R/D.$$

Then, by Lemma 6, it is sufficient to have k many rounds of the while loop such that $\left(\frac{1}{2}\right)^k \#\mathcal{C}_0 \leq \left(\frac{1}{2}\right)^k \frac{R}{D} < 1$. Hence, it is sufficient to have $k = \max\left\{\log_2\left(\frac{R}{D}\right) + 1, 0\right\}$ rounds. ◀

Then, if Algorithm 1 is implemented in such a way that $\mathcal{L}$ and $\mathcal{U}$ are pre-sorted (by submitters including the indices of their entry in the lists), the total on-chain running time of Algorithm 1 is $O\left(\#\text{submissions} \cdot \left(1 + \max\left\{\log_2\left(\frac{R}{D}\right) + 1, 0\right\}\right)\right)$, plus at most $\frac{R}{D}$ calls to $\mathcal{O}_B$.

7 User-calibrated precision

In this section, under idealized assumptions about the results produced by $\mathcal{O}_B$, we study the precision of the output of Algorithm 1. Particularly, we see that it depends on the lengths of the intervals submitted by the respondents.

▶ **Theorem 11.** *Suppose that the true value that Algorithm 1 is trying to determine is* v*, and that the underlying binary oracle* $\mathcal{O}_B$ *is always correct in determining whether a value is greater or less than* v*. Suppose that some respondent submits the interval* $I = (l_0^*, u_0^*)$ *such that* $v \in I$*. Furthermore, suppose that this respondent is willing to pay any required appeal fees in at most* $2\max\{\log_2(\#\mathcal{C}_0) + 1, 0\}$ *specifically chosen calls of* $\mathcal{O}_B$ *on behalf of claims that would be implied by* $v \in I$*. Then the response output by the oracle is in* I*.*

Proof. Consider the calls of $\mathcal{O}_B$ with respect to the values m_i that are considered through the various rounds of the while loop. Suppose for the moment that the respondent pays any appeal fees required when $m_i \notin I$ for rulings consistent with $v \in I$. Then, suppose we have passed through the while loop k times. We will iteratively define an interval I_k such that

$$\text{ultimate response} \in I_k \subseteq I.$$

We define these intervals as either of the form $I_k = (l_k^*, u_k^*)$ or the form $I_k = (l_k^*, u_k^*]$. We take $I_0 = I = (l_0^*, u_0^*)$ and then for $k > 0$:

$$I_k = \begin{cases} (l_{k-1}^*, m_k] & : \text{ if } \mathcal{O}_B \text{ rules that } v \leq m_k,\ m_k < u_{k-1}^* \\ I_{k-1} & : \text{ if } \mathcal{O}_B \text{ rules } v \leq m_k,\ m_k \geq u_{k-1}^* \\ (m_k, u_{k-1}^*) & : \text{ if } \mathcal{O}_B \text{ rules that } v > m_k,\ m_k \geq l_{k-1}^*,\ I_{k-1}^* = (l_{k-1}^*, u_{k-1}^*) \\ (m_k, u_{k-1}^*] & : \text{ if } \mathcal{O}_B \text{ rules that } v > m_k,\ m_k \geq l_{k-1}^*,\ I_{k-1}^* = (l_{k-1}^*, u_{k-1}^*] \\ I_{k-1} & : \text{ if } \mathcal{O}_B \text{ rules that } v > m_k,\ m_k \leq l_{k-1}^* \end{cases}$$

We note that in the first case, we must, in fact, have $l_{k-1}^* \leq m_k$, so I_k is well-defined. If l_{k-1}^* was a previous round m_i, then it must be a lower bound on $\mathcal{L} \cup \mathcal{U}$ in the kth round. Hence $l_{k-1}^* \leq m_k$. Otherwise, if $m_k < l_{k-1}^* = l_0^*$, this dispute is one in which we have assumed that the respondent is willing to pay appeal fees on behalf of $m_k < l_0^* < v$. However, this is incoherent with $\mathcal{O}_B$ ruling that $v \leq m_k$ by our assumptions on the correctness of $\mathcal{O}_B$. Similarly, we see I_k is, in fact, a non-empty interval in all cases, whose endpoints consist of elements of $\mathcal{L}$ and $\mathcal{U}$ in the kth round. Moreover, each $I_k \subseteq I_{k-1} \subseteq I$ by construction.

As the algorithm halts by Corollary 7, eventually, after w rounds of the while loop, all lower bounds in $\mathcal{L}$ will be (strictly) less than all upper bounds in $\mathcal{U}$ with output satisfying

$$l_j < \text{ultimate response} < u_i, \text{ for all } i, j.$$

In particular, the response is (strictly) between l_w^* and u_w^*, so

$$\text{ultimate response} \in I_w \subseteq I.$$

Finally, we show that there is at most one value in $\mathcal{C}_0$ which can arise as an $m_k \leq l_0^*$ for which the respondent would need to pay appeal fees in the kth round of the while loop. Instead suppose that $c_{k,1}, c_{k,2} \in \mathcal{C}_0$ with $c_{k,1}, c_{k,2} \leq l_0^*$ that could each arise as m_k in different executions. Suppose without loss of generality $c_{k,1} \leq c_{k,2}$. Take c_j to be the last common ancestor of these values in the binary search tree, then $c_{k,1} \leq c_j \leq c_{k,2} \leq l_0^*$. As $c_{k,1}$ and $c_{k,2}$ can arise as m_k, c_j can arise as m_j in its round. Then as the respondent is assumed to pay appeal fees for calls regarding $c_j \leq l_0^* < v$, $\mathcal{O}_B$ must rule that $v > c_j$. Hence $c_{k,1}$ cannot arise as a value of m_k. By Proposition 10 and repeating this argument for values $m_k > u_0^*$, the respondent need only pay appeal fees in at most $2 \max\{\log_2(\#\mathcal{C}_0) + 1, 0\}$ calls. ◀

A consequence of Theorem 11 is that if users require that the oracle output very precise values, they need only submit very precise interval estimates as respondents in which they are nonetheless confident that the true answer lies. Another consequence of Theorem 11, is that, again under idealized assumptions on $\mathcal{O}_B$, honest respondents will never be penalized.

▶ **Corollary 12.** *Suppose that the true value that Algorithm 1 is trying to determine is v, and that $\mathcal{O}_B$ is always correct in determining whether a value is greater or less than v. Suppose that a respondent submits the interval $I = (l_0^*, u_0^*)$ such that $v \in I$. Furthermore, suppose that this respondent is willing to pay all required appeal fees in $2 \max\{\log_2(\#\mathcal{C}_0) + 1, 0\}$ specifically chosen calls of $\mathcal{O}_B$ on behalf of claims that would be implied by $v \in I$. Then the user will not lose any appeal fees or deposits.*

Proof. By Theorem 11, the eventual value output by the procedure will be in I, hence the respondent will not lose his deposit. As discussed in the proof of Theorem 11, in the cases in which the respondent is assumed to contribute appeal fees (if necessary), he takes positions consistent with a true value of v and hence is always on the winning side. ◀

8 Equilibria in the respondent game

A key challenge of price oracles is that it is often unreasonable to speak about the price of an asset as being defined beyond a certain precision. An asset can be traded in many marketplaces simultaneously and while one might average together some weighted version of the prices in these different marketplaces, this will inevitably only give an approximation of the price. As a result, one might argue that there is some interval $(v-\epsilon, v+\epsilon)$ such that any element of this interval could be argued to be the price. Considering the transcendental nature of $\mathbb{R}$, one can expect this phenomenon to hold for other $\mathbb{R}$ or $\mathcal{S}$ valued oracles as well.

In previous sections, we have sometimes taken the idealized hypothesis that $\mathcal{O}_B$ rules "honestly" with respect to whether a given x is higher or lower than some single "true value." This might be a realistic (if optimistic) assumption when $x \notin (v-\epsilon, v+\epsilon)$. However, when presented with $x \in (v-\epsilon, v+\epsilon)$, it is more realistic to consider the choice made by $\mathcal{O}_B$ as being inevitably random, even when $\mathcal{O}_B$ is "honest."

In this section, we will consider a simplified model where the output of Algorithm 1 is distributed uniformly over $(v-\epsilon, v+\epsilon)$, and we will analyze respondent payoffs and incentives. Similar analysis using other distributions may be a subject for future work; however, already this simple model is not completely unreasonable. Heuristically, imagine that $\mathcal{O}_B$ responds "yes" and "no" with equal probability if posed the question "is x greater than the true value" for any $x \in (v-\epsilon, v+\epsilon)$. Further, suppose respondents placed intervals in such a way that $\mathcal{C}_0 \cap (v-\epsilon, v+\epsilon)$ consists of equally spaced points. Then, as the number of such points increases, the output's distribution becomes well approximated as uniform on $(v-\epsilon, v+\epsilon)$.

In this model, we examine conditions under which there is no economic incentive in terms of the payouts to coherent respondents for them to submit intervals smaller than 2ϵ. Of course, respondents may have some external interest in the output of Algorithm 1 such that they are incentivized to submit smaller intervals.

▶ **Proposition 13.** *Consider a model as described above, where the output value of Algorithm 1 is drawn uniformly from $(v-\epsilon, v+\epsilon)$ and payouts to correct respondents are given according to formula 1. There is an equilibrium where all respondents submit responses containing $(v-\epsilon, v+\epsilon)$; hence the game played by the respondents is Bayesian-Nash incentive-compatible. Moreover, suppose $\alpha^{2\epsilon} < e \approx 2.718$ and suppose that all respondents other than $\mathcal{USR}_i$ submit intervals that either include or do not intersect $(v-\epsilon, v+\epsilon)$. Then the respondent $\mathcal{USR}_i$ maximizes his expected payoff by submitting the interval $I_i = (v-\epsilon, v+\epsilon)$.*

Proof. The first claim is clear from the fact that rewards for respondents are paid from the lost deposits of other respondents. Now assume $\alpha^{2\epsilon} < e$. Suppose that a respondent $\mathcal{USR}_i$ submits an interval I_i such that $\text{length}(I_i) = \delta_i$ and $I_i \subseteq (v-\epsilon, v+\epsilon)$. Then, he has a $\frac{\delta_i}{2\epsilon}$ chance of being ruled correct and a $1 - \frac{\delta_i}{2\epsilon}$ chance of being ruled incorrect. Note that, as we assume that all other respondents submit intervals that either include or do not intersect $(v-\epsilon, v+\epsilon)$, the payoff for a response depends only on δ_i and whether $v_{\text{output}} \in I_i$. Hence,

$$E[\text{submit } I_i] = \frac{\#\text{ incorrect responses} \cdot D - \text{cost of first round } \mathcal{O}_B \text{ fees}}{\alpha^{-\delta_i} + \sum_{j \text{ such that } \mathcal{USR}_j \text{ correct}, j \neq i} \alpha^{-\text{length}(I_j)}} \cdot \alpha^{-\delta_i} \cdot \frac{\delta_i}{2\epsilon} - D \cdot \left(1 - \frac{\delta_i}{2\epsilon}\right).$$

However, the payoff for the honest strategy of submitting $I_i = (v - \epsilon, v + \epsilon)$ is given by

$$E[\text{honest}] = \frac{\#\text{ incorrect responses} \cdot D - \text{cost of first round } \mathcal{O}_B \text{ fees}}{\alpha^{-2\epsilon} + \sum_{j \text{ such that } \mathcal{USR}_j \text{ correct}, j \neq i} \alpha^{-\text{length}(I_j)}} \cdot \alpha^{-2\epsilon}.$$

Denote

$$A = \sum_{j \text{ such that } \mathcal{USR}_j \text{ correct}, j \neq i} \alpha^{-\text{length}(I_j)} \geq 0.$$

Then if we have

$$\frac{1}{\alpha^{-\delta_i} + A} \cdot \alpha^{-\delta_i} \cdot \frac{\delta_i}{2\epsilon} - \frac{1}{\alpha^{-2\epsilon} + A} \cdot \alpha^{-2\epsilon} \leq 0$$

$$\Leftrightarrow \left(1 + A\alpha^{2\epsilon}\right) \cdot \frac{\delta_i}{2\epsilon} - (1 + A\alpha^{\delta_i}) \leq 0,$$

that is sufficient to see that the honest strategy yields a higher expected payout.

However, $\frac{\delta_i}{2\epsilon} \in [0, 1]$, so we define

$$f(x) = xA\alpha^{2\epsilon} + x - A\alpha^{2\epsilon x} - 1$$

for $x \in [0, 1]$. Then

$$f'(x) = A\alpha^{2\epsilon} + 1 - A\alpha^{2\epsilon x} \cdot \ln\left(\alpha^{2\epsilon}\right) \geq A\alpha^{2\epsilon} + 1 - A\alpha^{2\epsilon} \cdot \ln\left(\alpha^{2\epsilon}\right) > 0$$

by the assumption that $\alpha^{2\epsilon} < e$. Then as $f(1) = 0$, one has that $f(x) \leq 0$ for all $x \in [0, 1]$. ◀

9 Conclusion

We have presented a completely crowd-sourced oracle for values in smart contracts from dense totally ordered sets that we expect to be particularly applicable as a price oracle. This proposal takes as an ingredient an oracle that can make binary decisions, for which one could use, in particular, the existing systems of Kleros, Augur, or ASTRAEA, then extending the influx of knowledge about the real world that they provide to a wider setting. The number of times the binary oracle must be called is limited to a reasonable bound in terms of the resources of parties who propose incoherent answers, not calling the system at all if all respondents submit mutually consistent answers. Hence the time required to compute this oracle should be suitable for many applications. Furthermore, the precision with which our proposed oracle returns its final answer is tuned to the precision of the most precise correct respondent so that the system can be as precise as its users require it to be.

References

1 Draft: fees in Kleros. https://github.com/kleros/research-docs/commit/e99053d5b81f6c8126352362a1cfe07f331033bd. Consulted: March 2019.

2 Ethereum white paper: A next-generation smart contract and decentralized application platform. https://github.com/ethereum/wiki/wiki/White-Paper. Consulted: April 2018.

3 Gnosis FAQ. https://gnosis.pm/faq. Consulted: April 2018.

4 Gnosis whitepaper. https://gnosis.pm/resources/default/pdf/gnosis-whitepaper-DEC2017.pdf. December 2017.

5 Technical Assessement of Augur. SportCrypt, Medium https://medium.com/@SportCrypt/technical-assessment-of-augur-5ff1a74df327. August 2018.

6 TLSnotary - a mechanism for independently audited https sessions. `https://tlsnotary.org/TLSNotary.pdf`. September 2014.

7 John Adler, Ryan Berryhill, Andreas G. Veneris, Zissis Poulos, Neil Veira, and Anastasia Kastania. ASTRAEA: A decentralized blockchain oracle. *2018 IEEE Confs on Internet of Things, Green Computing and Communications, Cyber, Physical and Social Computing, Smart Data, Blockchain, Computer and Information Technology, Congress on Cybermatics*, 2018.

8 Nadja Beneš. Getting to the Core: An Overview of our Contract Architecture. Gnosis blog `https://blog.gnosis.pm/getting-to-the-core-4db11a31c35f`. August 2017.

9 Vitalik Buterin. Ethereum and oracles. Ethereum Blog, `https://blog.ethereum.org/2014/07/22/ethereum-and-oracles/`. July 2014.

10 Vitalik Buterin. SchellingCoin: A minimal-trust universal data feed. Ethereum Blog: `https://blog.ethereum.org/2014/03/28/schellingcoin-a-minimal-trust-universal-data-feed/`. March 2014.

11 Lin William Cong, Zhiguo He, and Jingtao Zheng. Blockchain Disruption and Smart Contracts. *SSRN Electronic Journal*, January 2017. `doi:10.2139/ssrn.2985764`.

12 Mike Dalton. Chainlink acquires Town Crier, a hardware-based oracle. Unhashed `https://unhashed.com/cryptocurrency-news/chainlink-acquires-town-crier-hardware-based-oracle/`. November 2018.

13 Michael del Castillo. Thomson Reuters to power blockchain contracts with experimental service. Coindesk, `https://www.coindesk.com/thomson-reuters-power-blockchain-contracts-new-experimental-service/`. June 2017.

14 Steve Ellis, Ari Juels, and Sergey Nazarov. ChainLink: A decentralized oracle network. `https://link.smartcontract.com/whitepaper`. September 2017.

15 Clément Lesaege and Federico Ast. Kleros: Short paper v1.0.5. `https://kleros.io/assets/whitepaper.pdf`. January 2018.

16 The Maker Team. The Dai Stablecoin System. `https://makerdao.com/whitepaper/DaiDec17WP.pdf`. December 2017.

17 Jack Peterson and Joseph Krug. Augur: a decentralized, open-source platform for prediction markets. `https://bravenewcoin.com/assets/Whitepapers/Augur-A-Decentralized-Open-Source-Platform-for-Prediction-Markets.pdf`. July 2018.

18 Thomas Schelling. *The Strategy of Conflict*. Harvard University Press, 1980. URL: `https://books.google.ca/books?id=7RkL4Z8Yg5AC`.

19 Paul Sztorc. Truthcoin: Peer-to-peer oracle system and prediction marketplace. `https://www.truthcoin.info/papers/truthcoin-whitepaper.pdf`. Version 1.5, December 2015.

20 Katrin Tinn. Blockchain and the future of optimal financing contracts. In *European Finance Association 45th Meeting*, EFA 2018. European Finance Association (EFA), 2018. `doi:10.2139/ssrn.3072854`.

21 Fan Zhang, Ethan Cecchetti, Kyle Croman, Ari Juels, and Elaine Shi. Town Crier: An authenticated data feed for smart contracts. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 270–282, New York, NY, USA, 2016. ACM. `doi:10.1145/2976749.2978326`.

Cryptocurrency Egalitarianism: A Quantitative Approach

Dimitris Karakostas
University of Edinburgh, UK
IOHK, Hong Kong
dimitris.karakostas@ed.ac.uk

Aggelos Kiayias
University of Edinburgh, UK
IOHK, Hong Kong
akiayias@inf.ed.ac.uk

Christos Nasikas
University of Athens, "Athena" Research Center, Greece
xnasikas@di.uoa.gr

Dionysis Zindros[1]
University of Athens, Greece
IOHK, Hong Kong
dionyziz@di.uoa.gr

Abstract

Since the invention of Bitcoin one decade ago, numerous cryptocurrencies have sprung into existence. Among these, proof-of-work is the most common mechanism for achieving consensus, whilst a number of coins have adopted "ASIC-resistance" as a desirable property, claiming to be more "egalitarian," where egalitarianism refers to the power of each coin to participate in the creation of new coins. While proof-of-work consensus dominates the space, several new cryptocurrencies employ alternative consensus, such as proof-of-stake in which block minting opportunities are based on monetary ownership. A core criticism of proof-of-stake revolves around it being less egalitarian by making the rich richer, as opposed to proof-of-work in which everyone can contribute equally according to their computational power. In this paper, we give the first quantitative definition of a cryptocurrency's *egalitarianism*. Based on our definition, we measure the egalitarianism of popular cryptocurrencies that (may or may not) employ ASIC-resistance, among them Bitcoin, Ethereum, Litecoin, and Monero. Our simulations show, as expected, that ASIC-resistance increases a cryptocurrency's egalitarianism. We also measure the egalitarianism of a stake-based protocol, Ouroboros, and a hybrid proof-of-stake/proof-of-work cryptocurrency, Decred. We show that stake-based cryptocurrencies, under correctly selected parameters, can be perfectly egalitarian, perhaps contradicting folklore belief.

2012 ACM Subject Classification Security and privacy → Economics of security and privacy

Keywords and phrases blockchain, egalitarianism, cryptocurrency, economics, proof-of-work, proof-of-stake

Digital Object Identifier 10.4230/OASIcs.Tokenomics.2019.7

Funding Research partially supported by H2020 projects PRIVILEDGE #780477 and MHMD #732907.

[1] Corresponding author

International Conference on Blockchain Economics, Security and Protocols (Tokenomics 2019).
Editors: Vincent Danos, Maurice Herlihy, Maria Potop-Butucaru, Julien Prat, and Sara Tucci-Piergiovanni;
Article No. 7; pp. 7:1–7:21

OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

In 2008, Satoshi Nakamoto proposed Bitcoin [24], the first and most successful cryptocurrency to date. Bitcoin introduced a cryptographic consensus protocol in which transactions are organized into blocks which are put in a globally agreed sequence, the *blockchain*, despite the presence of adversaries and without the need of any setup or identity system. Since its inception, a plethora of alternative cryptocurrencies, or "altcoins," have sprung into existence, each claiming its own features.

A major thread of research has focused on the mandates of block generation, specifically the mechanism of identifying the party responsible for producing a new block at any point. Bitcoin, as well as the majority of altcoins, employs *proof-of-work* [11], where block generation is called *mining* and blocks are produced by *miners* who expend computational power to solve cryptographic puzzles. On the other hand, the most prominent alternative mechanism is *proof-of-stake*. In proof-of-stake, block generation is called *minting* and blocks are produced by *minters* who "stake" their coins, i.e., users who own a set of coins and use them to participate in the consensus protocol. Intuitively, in both cases a leader is drawn at regular intervals at random from the block generators population, with a probability of selection proportional to their computational power or stake respectively.

Block generators are incentivized to produce blocks by receiving a *reward* for each block they successfully produce and which is subsequently adopted in the resulting blockchain. In many cryptocurrencies, the rewards serve a dual purpose: incentivise the the miners/minters but also create and distribute the underlying cryptocurrency to the system's maintainers. Taking this into account, in this paper, we consider the block generators as investors and focus on the comparison of the *expected* returns of investors with different purchasing power. The central economic property which arises is that of cryptocurrency *egalitarianism*. In an ideal world, investing a certain amount of capital to produce blocks should result in rewards proportional to that capital; that is, both a *poor* investor and a *rich* investor should receive returns in proportion to their investment in expectation. In this point of view, wealthy investors should not be rewarded with disproportionate rewards and everybody should have equal opportunity to both participate and earn rewards. As we will see, this is far from true with most cryptocurrencies today.

Until now, the term *egalitarianism* has been left undefined, although several cryptocurrencies claim to be more egalitarian than others [30] [23]. However, lacking a quantifiable metric, the question of whether some cryptocurrencies are more *egalitarian* than others remains ill posed. Our paper aims at putting forth the first concrete definition of egalitarianism, in a way which is generic and can be applied to any cryptocurrency. Our definition provides a metric, which can be practically measured and used to compare different cryptocurrencies. Using our model, we measure the egalitarianism of four indicative proof-of-work–based cryptocurrencies: Bitcoin, Litecoin [21], Ethereum [6, 31], and Monero [30]. Bitcoin, being the first and most successful cryptocurrency to date, was chosen as the baseline of comparison. Ethereum is the most promising altcoin and is currently the largest decentralized cryptocurrency by market cap after Bitcoin[2]. Litecoin and Monero, although not next by market cap, make claims [30, 23] of increased egalitarianism because of their design. We assess their claims and find them in agreement with our data, thus presenting for the first time economic comparisons which quantify them precisely. On the pure proof-of-stake side, as will soon become clear,

[2] All references to market cap in this paper are made according to `https://coinmarketcap.com` [January 2019].

egalitarian behavior is similar across all coins independently of externalities such as hardware characteristics. Therefore, it suffices to perform a case study of an indicative proof-of-stake protocol. We study the case of pure proof-of-stake, applied on a protocol consistent with Ouroboros [19], as well as a hybrid proof-of-work/proof-of-stake cryptocurrency, Decred [9]. We find that, in an open market, pure proof-of-stake coins can be perfectly egalitarian, contrary to their proof-of-work counterparts. However, we note that variations of proof-of-stake, such as "delegated proof-of-stake," may not be perfectly egalitarian, since the delegates, i.e., the leaders of the stake pools which are formed, typically earn extra profits for managing the stake pools [5].

Our Contributions and Roadmap. This work provides a quantitative evaluation of cryptocurrency egalitarianism. To the best of our knowledge this is the first work to provide a treatment of this property and acts as the foundation for comparing cryptocurrency fairness when it comes to reward distribution. Specifically, the contributions of our research are summarized as follows:

1. We define an exact measure of cryptocurrency *egalitarianism*; to do this, we first define the *egalitarian curve* of a cryptocurrency from which we extract the measure.
2. We measure and compare the egalitarian curve and egalitarianism of four indicative proof-of-work cryptocurrencies (Bitcoin, Ethereum, Litecoin, Monero), one representative proof-of-stake protocol (Ouroboros), and a hybrid cryptocurrency (Decred), using current market data.
3. We show that proof-of-stake, when correctly parameterized, is, perhaps unexpectedly, perfectly egalitarian.

The rest of this paper is structured as follows. We begin by reviewing related work and preliminaries in Sections 2 and 3. Next, we put forth our definition for the egalitarian curve and egalitarianism of a cryptocurrency and motivate its intuition in Section 4. In Section 5 we present empirical data for several cryptocurrencies of interest and evaluate them under our model, in order to deduce whether previous intuitive claims are indeed correct. Finally, the conclusions of our research are drawn in Section 6.

2 Related work

The macro and microeconomics of blockchain design have been studied from several perspectives but remain an active area of research with a number of open questions. Incentives for block generation according to the honest protocol have been explored for both proof-of-work and proof-of-stake.

Proof-of-work protocols such as Bitcoin were formalized in the Bitcoin Backbone [14, 15] papers and follow-up works [25]. The seminal work of Selfish Mining [12, 28] showed that the honest behavior is not incentive-compatible in Bitcoin, but the protocol can be modified to behave that way. However, in restricted models, Bitcoin can be shown to be incentive-compatible [18]. Proof-of-stake protocols such as Ouroboros [19] can be designed from the ground up to be incentive-compatible. The question of how to incentivize parties to conduct pool formation into the desired number of pools, or groups of minters, was studied in [5].

The above works study the incentives of blockchain systems from the designer's point of view so that participants do not deviate from the prescribed protocol. A related question is how *fair* the protocol is to participants themselves, and in particular to honest participants. The Backbone and Selfish Mining works include attacks in which an adversary can strategically

harm *chain quality*, causing the number of blocks and, in turn, the respective rewards, to be disproportionate to their contributed computational power, thereby harming fairness against honest participants. Fruitchains [26] proposes a protocol which solves this problem. In these works, handing out rewards in exact proportion to computational power is considered "fair."

Egalitarianism, in the way considered in the paper, has been studied in proof-of-work systems from a technological point of view with respect to *memory-hard functions* in [2, 3]. However, the question of whether computational power grows proportionally to capital invested, i.e., whether larger wealth results in more than proportional rewards, has not been previously studied. Therefore, our work aims at filling this gap by studying the effects of economies of scale when applied to cryptocurrency generation.

Equitability of cryptocurrencies. Fanti *et al.* analyze economic blockchain fairness in [13], where they define *equitability*. They study the evolution of a system after a series of rounds, putting forth the property that stake ownership remains in proportion *before* and *after* rewards have been awarded. By studying the behaviour of the returns' *variance* under the randomness of executions, they show that the distribution of capital follows a Pólya process. Our work augments their results by quantifying the *expectation* of rewards and then studying the variance under the randomness of initial capital allocation. In our work, we show that computational power is not proportional to the invested capital, and hence the analogy between proof-of-work computational power and proof-of-stake capital breaks down, and a more detailed study is needed. Additionally, we remark that proof-of-work miners also reinvest their proceeds in the mining operation, albeit slowly, as proof-of-stake minters do. For example, empirical data show that large-scale miners pay for electricity using their proceeds [17]. Hence, both mining and minting follow Pólya processes as modelled by their paper. Regardless, *egalitarianism* and *equitability* are orthogonal. A cryptocurrency can be perfectly egalitarian and poorly equitable and vice versa. It is possible to obtain a cryptocurrency both egalitarian and equitable by adopting correctly parameterized proof-of-stake under a geometric reward function.

3 Preliminaries

Before studying the egalitarianism of different cryptocurrency consensus mechanisms, we provide a description of the leader election process, which is a central part of each blockchain consensus mechanism. We give an overview of the details of the two most common decentralized consensus mechanisms, proof-of-work and proof-of-stake, in order to establish an understanding of the differences in egalitarianism between the two models.

Proof-of-work. The core idea behind proof-of-work cryptocurrencies is solving the proof-of-work inequality. Specifically, the mining hardware is provided with two constants, PREVID and DATA, i.e., the id of the tip of the adopted blockchain and the data which need to be appended to it. The mining device then *brute-force* searches for some string NONCE, such that $\mathsf{H}(\text{PREVID}||\text{DATA}||\text{NONCE}) \leq T$ for some hash function H defined by the system. Here, T is a – relatively – small number called the *difficulty target*, which is adjusted in order to ensure a stable block production rate, although typically remains constant for periods of consecutive blocks called *epochs* – for example, in Bitcoin, epochs are 2016 blocks long [4]. Because the search for solutions is brute-forced, the expected number of solutions found by a given miner is proportional to the number of evaluations of the hash function H she can obtain in a given time frame.

The number of hash evaluations is one of the several critical parameters to consider when purchasing mining hardware. Other important parameters include the price of a mining unit, as well as its electricity consumption. Mining hardware is divided in various tiers based on performance, namely CPU miners, GPU miners, FPGA miners, and specialized ASIC miners [29]. Although the pricing of such devices may be similar, the hashing rate and, in turn, the return on investment, is highly dependent on the hardware's tier. For example, the mining hardware "Whatsminer M10" produced by the company "MicroBT" costs $1,022.00 per unit and produces $0.104266 per hour of operation in net gains, i.e., average mined Bitcoins per hour denominated in US dollars with today's prices (December 2018) minus the electricity costs. On the other hand, the mining hardware "8 Nano Pro" produced by the company "ASICMiner" costs $6,000.00 per unit, but produces $0.315327 per hour of operation in net gains, i.e., almost three times the hourly net gains of its cheaper competitor. Thus, if one can afford to purchase the more expensive hardware, each of their subsequent dollar invested in electricity returns more mined coins.

It has long been folklore knowledge in the blockchain community that mining becomes more egalitarian by using a memory-hard proof-of-work function. This intuition is correct, the core reason being the difficulty to construct specialized hardware for memory-hard functions. For example, no ASICs currently exist for Monero mining. Therefore, the only way to scale mining operations is by purchasing more hardware. However, since the mining hardware in this case varies little, both in terms of cost and performance, scaling returns become proportional to investments. To the best of our knowledge, this paper is the first to confirm this correspondence between the memory-hardness of proof-of-work hash functions and the economics of mining.

▶ Remark 1 (Block generation at scale). We only analyze the scaling of the economics of mining with respect to hardware. We also do not take into account basic costs such as shipping and the availability of a basic machine to co-ordinate mining (such as a personal computer not performing mining itself). A multitude of additional factors play important roles for mining operations, such as space rental costs, machine cooling and maintenance costs, as well as bulk electricity purchase. As is common in economies of scale, these relative costs are reduced for large-scale operations, although they are similar for all proof-of-work cryptocurrencies and thus do not affect relative comparisons between them. We also remark that we analyze mining costs for small capital investments. If larger capital, e.g. above a few million US dollars, is available, corporations can develop their own specialized hardware and gain a competitive advantage by treating it as a trade secret [29]. Indeed, these details make our comparison in favour of proof-of-stake *more pronounced*, as proof-of-stake operations do not incur such types of costs and do not lend themselves to specialized mining hardware research. We leave the analysis and calculation of egalitarianism under these parameters for future work.

Proof-of-stake. In proof-of-stake, a minter is selected in proportion to the stake they hold, which is to say proportionally to the amount of money they own. There exist a number of flavours of this process. In one case, all coins automatically participate in the leader election process – this is the case for Ouroboros [19] and Ethereum's Casper [7]. In a second flavour, the stake has to *opt-in* to participate in the election by a special process, such as purchasing a *ticket* or becoming a delegate of the stake of other users. This is the case for cryptocurrencies such as Decred [9] and EOS [20]. Among those participating in the election, a leader is elected at random, in proportion to their stake.

Proof-of-stake is often criticized for its lack of egalitarianism. The rationale is that, in proof-of-stake, the more money one stakes, the more money one generates. Thus, the *rich get richer*, which is precisely the *opposite* of egalitarianism. Additionally, in proof-of-stake systems, the money owners could constitute a *closed, rich club*, refusing to share the assets with any outsiders. In contrast, this argument claims, proof-of-work is naturally egalitarian: everyone is paid not according to the money they own, but according to the computational power they put to work. In this case, since computational power is a *natural* thing and cannot be exclusively owned, a closed rich club cannot be formed. Although this argument seems agreeable at first, the results of our work contradict it. In fact, correctly parameterized stake-based systems are much more egalitarian than work-based ones.

It is instructive to dispel the above argument intuitively, before we support our position with data. Firstly, the argument that money can be exclusively owned, but computational power cannot, is misguided. Indeed, this may be true in the case of a peculiar oligopoly, where a small faction of parties mutually agrees to never sell to outsiders, despite external demand. However, in an open market, both money and computational power can be freely purchased and, in fact, any non-negligible amount of computational power must be necessarily purchased that way. In the present work, **we assume an open market** for both mining hardware and financial capital which allows participation in the respective systems. Therefore, given that both money and computational power are purchasable, we now need to consider the funds one needs to invest either in technology or in financial capital in order to maximize the returns from a cryptocurrency's block generation mechanisms. The amount of cryptocurrency generated by a given investment can be concretely measured and compared, thus the question can now be analyzed quantitatively and answered concretely.

We should note that variations of proof-of-stake, such as "delegated proof-of-stake," may not be perfectly egalitarian, since the delegates, i.e., the leaders of the stake pools which are formed, typically earn extra profits for managing the stake pools [5]. In this paper, we only concern ourselves with non-delegated variants, i.e., *pure* proof-of-stake protocols. We leave the study of the contrast between pool formation mechanism truthfulness (or Sybil-resilience) and egalitarianism for future work.

4 Defining egalitarianism

Having established the basics of consensus mechanisms, we now propose the first definition of an economic measure of *egalitarianism* in cryptocurrencies. Before we present our definition, let us first state the *desiderata* of such a definition. First of all, we want to allow concrete measurements to be performed on cryptocurrencies and data to be extracted in a manner that is quantitative and not vague. Thus far, the claims for egalitarianism in various cryptocurrencies have been hand wavy, using a rhetoric which fails to include exact data [30, 23]. As such, different cryptocurrencies claim egalitarianism over the others, without demonstrating the claims or provide conclusive arguments. Secondly, a definition of egalitarianism must measure the block generation returns of a "rich dollar" compared to that of a "poor dollar." We thus desire a measure which, for a particular cryptocurrency, extracts a smaller value to indicate a *lack of egalitarianism* (e.g. a case where large wealth generates blocks disproportionately faster than small wealth) and a larger value to indicate *perfect egalitarianism* (where every invested dollar has exactly equal power in terms of cryptocurrency generation).

As a means towards establishing our egalitarianism definition, we define the *egalitarian curve* f of a cryptocurrency. The horizontal axis of this curve plots the financial capital which is available for investment denominated in a fiat currency, USD.[3] The vertical axis plots the

[3] Given that we explore a small investment duration, it makes little difference whether these are nominal

Return On Investment (ROI), which measures the cryptocurrency amount that is freshly generated in the investment period and remains unspent at the end of the investment period, given an optimal allocation of the initial capital. We require the Return On Investment is necessarily *freshly generated* cryptocurrency; thus, it must be newly mined or minted, and not part of the initial capital. Of course, purchasing cryptocurrency which has already been generated is an investment option, but it is immaterial to our egalitarianism definition, which focuses on measuring the egalitarianism of freshly generated cryptocurrency. Finally, the curve is plotted with a fixed investment duration in mind – in this paper, we use a duration of 1 year. Naturally, curves of different cryptocurrencies can be compared only if they use the same duration.

▶ **Definition 2** (Egalitarian curve). *Given a cryptocurrency c, an investment period interval d, the set of all possible investment strategies $\mathcal{B}$, we define the* egalitarian curve $f_{c,d} : \mathbb{R}^+ \longrightarrow \mathbb{R}^+$ *of c for investment period d as:*

$$f_{c,d}(v) = \frac{\max_{B \in \mathcal{B}} \mathbb{E}[B(v)] - v}{v}$$

The value $\max_{B \in \mathcal{B}} \mathbb{E}[B(v)]$ identifies the maximum expectation of returns across all investment strategies $\mathcal{B}$, i.e., the amount of returns which the *optimal* strategy ensures for a given initial capital v. The expectation is taken with the blockchain execution as a random variable, since returns vary by execution (the randomness of the execution can affect the returns of the strategy, as the same strategy can bring larger returns if the participant is "lucky" and happens to produce many blocks [13]).

We remark that we do allow strategies to reinvest capital. For instance, returns earned from mining rewards can be reinvested in electricity costs for future mining. Furthermore, for unit consistency, we assume the strategy $B(v)$ returns the freshly generated coins denominated in the same units as the capital v was given in, such that f represents a ROI; thus, we denominate the generated cryptocurrencies in USD using the market exchange rate.

It is now straightforward to define the *ideal egalitarian curve*. In this case, the ROI is stable regardless of capital invested. Under these ideal conditions, the amount of freshly generated cryptocurrency is exactly proportional to the money invested. Thus, the ideal curve is any constant curve.

As an interesting thought experiment, consider the egalitarian curve which is decreasing. In this case, the poor would receive proportionally more newly created cryptocurrencies for every dollar they invest, i.e., it would be a redistribution of wealth from the rich to the poor. However, one can quickly see that, in decentralized cryptocurrencies where the identities of the participants are unknown, it is impossible to hope for something better than the constant curve. Indeed, the fact that decentralized cryptocurrencies allow anonymous generation of new identities [10] allows a rich investor to split their investment into smaller ones. Thus, if the curve were ever to have a negative slope, the sum of the smaller splits of the rich investment would achieve a higher gain. By the definition of the curve, which mandates that it depicts the ROI of an *optimal* investment, this would be a contradiction. The following lemma makes the above intuition more precise:

▶ **Lemma 3** (Sybil strategies). *Fix a cryptocurrency c and an investment period interval d. Given capital v, for every natural number $i \in \mathbb{N}^\star$, it holds that $f_{c,d}(v) \leq f_{c,d}(i \cdot v)$.*

USD or real USD, as long as they are the same when applying comparisons.

The proof of this Lemma is available in Appendix A.

Using our definition of the egalitarian curve, we now define egalitarianism as a concrete number. We begin by considering the initial capital v as a random variable following a certain distribution $\mathcal{D}$. Egalitarianism is defined as the variance of the expected ROI when the capital is chosen from the given distribution.

▶ **Definition 4** (Egalitarianism). *Given a cryptocurrency c, an investment period duration d and an initial capital distribution $\mathcal{D}$, we define the* egalitarianism *e of c for investment duration d under initial capital distribution $\mathcal{D}$ as follows:*

$$e_{c,d,\mathcal{D}} = -\mathit{Var}_{v \leftarrow \mathcal{D}}[f_{c,d}(v)]$$

where f is the egalitarian curve of c.

The intuition behind this definition is that, to have egalitarianism, the ROI must remain the same across different capital investments. As such, any deviation from the mean is non-egalitarian. Naturally, if the egalitarianism of a certain cryptocurrency is *higher* than another's, we say that the former is *more egalitarian* than the latter. Of course, to be accurate, such comparisons must only be made after fixing the parameters c and d as well as the initial capital distribution $\mathcal{D}$. We will now fix the distribution $\mathcal{D}$ to be the uniform distribution between a minimum and a maximum capital. This choice corresponds to the intuition that the returns are the same for all initial capitals alike. Clearly a cryptocurrency with an ideal egalitarian curve is perfectly egalitarian, as we now define.

▶ **Definition 5** (Perfect egalitarianism). *A cryptocurrency c is* perfectly egalitarian *for investment duration d and initial capital distribution $\mathcal{D}$ if $e_{c,d,\mathcal{D}} = 0$.*

5 Experimental results

Having established our theoretical framework, we now provide experimental results on the egalitarianism of various cryptocurrencies. Our experiments utilize the *egalitarian curve* definition of Section 4 in order to concretely confirm – or disprove – the egalitarianism claims of some of the major, both proof-of-work and proof-of-stake, cryptocurrencies.

In conducting our experiments **we assume a static environment**. Specifically, we assume that the token prices, as well as the distribution of funds which are available for purchasing mining hardware are static and follow the snapshot of the world which we took at the time of writing. Furthermore, we assume that our mining operation would not substantially affect these parameters if it were to be applied on this environment. Finally, we assume that the set of available strategies $\mathcal{B}$ comprises of the honest strategies, e.g. not including selfish mining which could provide better ROI by diverging from the protocol.

Proof-of-work. We have experimentally analyzed the egalitarianism of the following proof-of-work coins: Bitcoin, Litecoin, Ethereum, and Monero. These cryptocurrencies act as a representative sample among the thousands of existing cryptocurrencies. Bitcoin is the largest and most successful cryptocurrency by market cap. Litecoin is the first cryptocurrency aimed at becoming more egalitarian by replacing Bitcoin's SHA256 work function with scrypt [27], a more memory-hard function. Ethereum is one of the most promising alternative cryptocurrencies, the first to support smart contracts, and the second largest by market cap; its work function is different from both Bitcoin and Litecoin. Finally, Monero is special with claims of strong egalitarianism due to its memory-hard mining function, Cryptonight [30]. Furthermore, its protocol is often updated to maintain egalitarianism [8].

Table 1 A list of the parameters used in our proof-of-work mining simulations. Some parameters are system-agnostic, whereas others depend on the cryptocurrency c.

Variable	Description	Unit	BTC	ETH	LTC	XMR	DCR
$\|d\|$	duration of investment	years	1				
EC	electricity cost	USD / kWh	0.08				
BGR(c)	block generation rate	blocks / s	1 / 600	1 / 14.7	1 / 150	1 / 120	1 / 298
THR(c)	total hash rate	Thash / s	34,727,437	179.50374	174.537	0.00033859	178,760
BR(c)	reward per block	tokens	12.5	3	25	3.37	11.38
TP(c)	token price	token / USD	4,074.25	126.12	32.10	47.27	18.62

As expected, our experiments show that Bitcoin is the least egalitarian of the four, with Ethereum following next. Monero is more egalitarian than both, with Litecoin being the most egalitarian among the proof-of-work coins we have studied.

For our experimental setting, we worked as follows. First, we collected empirical data which describe the available mining hardware options in the market. For each machine choice, we determined the cost of investment. This is comprised of its initial price (in USD) as well as its energy cost of operation (in Watts). The cost of operation was translated to USD per hour by considering the electricity cost of KWh. As a reference, we used the lowest average KWh cost in the United States, i.e., \$0.08 per KWh [1]. This reference electricity cost is an estimation which can vary depending on the country of operation.

Second, we use the reported hash rate of each mining hardware machine to extract an expectation of the freshly mined coins it would generate per hour, if it were to run continuously. This expectation is taken over the randomness of all honest blockchain protocol executions. As such, each party is awarded block rewards in proportion to their computational power. The difference between revenue per unit of time and cost of operation per unit of time produces an *income rate*, which is measured in USD per hour. For our experiments, we use an interval of investment with $|d| = 1$ year. Although this choice is arbitrary, it corresponds to the usual definition of ROI in traditional finance.

Our investment strategy is as follows. The initial available capital is allocated to an upfront technology investment, in which an integer instance of the Unbounded Knapsack problem [22] is solved using dynamic programming[4] to optimize the total cash flow. Subsequently, as long as the cash flow is positive, the purchased machines operate for the indicated total duration, reinvesting part of the freshly minted coins in electricity costs, in order to generate more coins. Eventually, this strategy produces an income of freshly generated coins, which have not been spent and are reported as the strategy's income.

To calculate our concrete numbers, we employ the constants shown in Table 1. We use the expected block generation rates for each cryptocurrency, as well as the reward per block, token price, and mining difficulty at the time of writing, all of which we assume remain constant. The variance of electricity cost, the duration of investment, as well as small fluctuations in price and difficulty do not qualitatively change the shape of our egalitarian curves (see Appendix C).

Let $\mathcal{M}$ denote the set of all available mining machines. For each machine $m \in \mathcal{M}$, our empirically collected data specifies the following parameters:

(i) the energy consumption rate ECR(m) in Watts,
(ii) an initial cost of purchase IC(m) in USD, and
(iii) a hash rate HR(m) in Terahashes per second.

[4] The source code of our implementation for this calculation is available in our repository.

Given the above, we can now calculate the expected income rate per hour $\mathbb{E}[\text{IR}(m)]$ for a given machine m and a cryptocurrency c. In the following equation, the first part identifies the income per hour, i.e., the amount of tokens (denominated in USD) which the machine produces per hour, whereas the second part of the equation identifies the electricity cost, i.e., the product of the consumed electricity multiplied by the price of a single KWh:

$$\mathbb{E}[\text{IR}(m)] = 3600 \cdot \frac{\text{HR}(m)}{\text{THR}(c)} \cdot \text{BR}(c) \cdot \text{BGR}(c) \cdot \text{TP}(c) - \text{ECR}(m) \cdot \text{EC}$$

There are many possible configurations for technology investments. Each configuration comprises of a number of copies $n \in \mathbb{N}$ of every machine type $m \in \mathcal{M}$. Therefore, we define each configuration as $\overline{m} \subseteq \mathcal{M} \times \mathbb{N}$, with total initial cost of investment for such configuration being $\text{IC}(\overline{m}) = \sum_{(m,n)\in\overline{m}} n \cdot \text{IC}(m)$.

The above figure is given in USD per hour and, since the initial capital should suffice to buy the machines of the configuration, we require that $\text{IC}(\overline{m}) \leq v$, where v is the initial available capital at the beginning of the simulation.

Now, in order to identify the strategy's optimal net income for the interval d, we iterate over all possible machine configurations, for which the above inequality holds, and choose the one with the maximum returns:

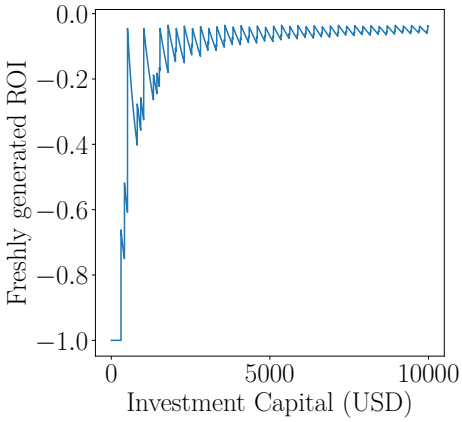
$$B_{\text{OPT}}(v) = \max\{ \sum_{(m,n)\in\overline{m}} |d|\mathbb{E}[\text{IR}(m)] : \overline{m} \subseteq \mathcal{M} \times \mathbb{N} \wedge \text{IC}(\overline{m}) \leq v\}$$

We note that this is only an approximation to the optimal (in our limited model) solution, which we used in our simulations. We consider this sufficiently close to optimal to allow for the calculation of egalitarianism. We give an integer programming formulation of the optimal strategy for capital allocation in Appendix B. We remark here that the general problem of mining hardware allocation (including our simplified approximation) is computationally hard [16], as both the Knapsack and the Integer Programming problems are NP-complete.

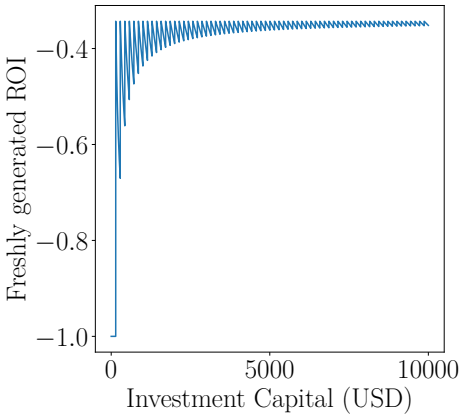
As the simulation parameters are many and diverse, in order to allow others to run the experiments with different values, as well as for reasons of reproducibility and falsifiability, we openly release our mining investment optimizer as well as our data for public use[5].

The egalitarianism of Bitcoin, Ethereum, Litecoin and Monero are shown in Figures 1a, 1b, 1c, and 1d respectively. Decred is a hybrid proof-of-work/proof-of-stake cryptocurrency, in which block generation is a collaboration between miners and minters. Specifically, each block which is mined via proof-of-work needs to be "vouched for" by a certain number of minters, who give it a vote of confidence. Both the miners and the minters who participate in block generation are rewarded. An investor can therefore choose to participate in Decred by either investing in mining hardware and performing proof-of-work, or by purchasing stake and performing proof-of-stake (or a combination thereof). We note that the choice of whether to mine or mint Decred is not always clear. While mining may be more profitable for a certain initial capital, it can also carry various risks. For instance, if the difficulty increases, the mining hardware may be rendered inefficient and also hard to sell. Proof-of-work also carries the operational overhead discussed in Remark 1. On the other hand, stake can always be sold, although the price may fluctuate, and carries negligible operational overhead. As the decision between the two is not obvious, we analyze both strategies independently. The egalitarianism of proof-of-work mining for Decred is shown in Figure 1d.

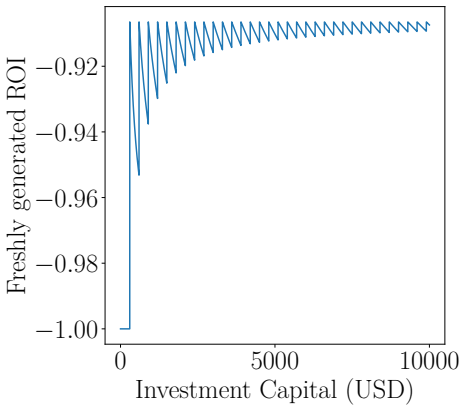
[5] Our mining investment calculator and our mining hardware data are available under the MIT license and a Creative Commons 4.0 Attribution License respectively at `https://github.com/decrypto-org/egalitarianism`.



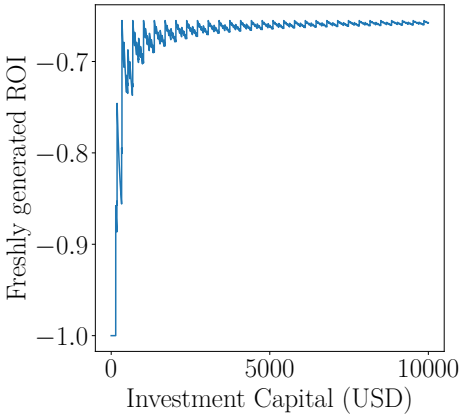
(a) Bitcoin



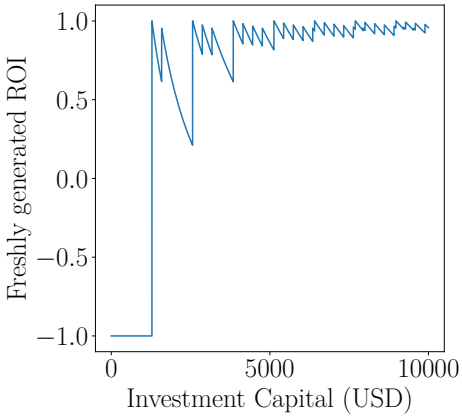
(b) Ethereum



(c) Litecoin

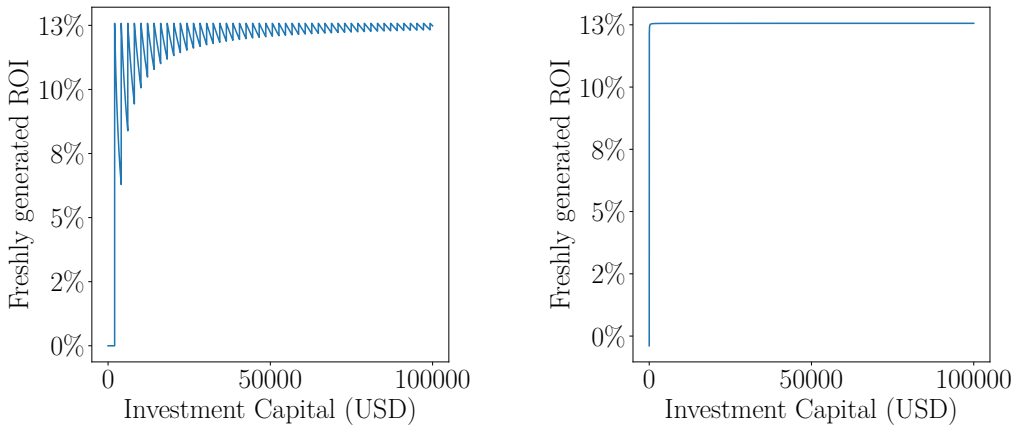


(d) Monero



(e) *Proof-of-work* Decred

Figure 1 Egalitarianism curves of the proof-of-work cryptocurrencies analyzed in this work.



(a) *Proof-of-stake* Decred **(b)** *Pure* proof-of-stake Ouroboros

Figure 2 The egalitarianism curves of the proof-of-stake systems analyzed in this work.

It is evident from all figures that the ROI is "capped" by a maximum value, which is observed in specified intervals. Indeed, this value identifies the ROI of the *best available* machine and is in line with Lemma 3. In other words, as long as an investor is able to buy the machine which returns the most profits, then they achieve the best possible ROI. In case an investor does not have enough capital to buy the best mining product, they may buy a less profitable machine and achieve less, though still positive, ROI. This observation explains the small spikes in ROI which may be seen e.g. in Bitcoin's figure for capital in the range $[0, 2000]$. Also, in case the capital is *more* than the cost of the machine, then the remaining capital is effectively discarded. Therefore, although two investors A, B may start with initial capital $v_A < v_B$, if their returns, in absolute terms, are the same, then the ROI of B will be smaller as a percentage compared to the ROI of A. This observation explains the decrease in ROI after the spikes. Finally, we observe that, as the capital increases, the ROI converges to the cap. This is explained by the fact that the "discarded" capital, i.e., the capital which cannot be invested in mining hardware, is a significantly smaller percentage of the total capital for large investments.

Proof-of-stake. We now analyze the proof-of-stake egalitarianism in two settings. First, we consider pure proof-of-stake, which can be applied on top of a protocol like Ouroboros. In this case, *pure* is in opposition to *delegated* proof-of-stake, a setting where the stakeholders are required to delegate their stake to other parties, namely "stake pools" and is deployed in cryptocurrencies such as EOS, Bitshares, and others. Second, we consider the case of minting Decred via its proof-of-stake mechanism.

The egalitarian curve for *staking* Decred is illustrated in Figure 2a. As mentioned above, Decred is an opt-in staking cryptocurrency, where staking occurs by purchasing so-called *tickets*. Since the price of a ticket is quantized, egalitarianism is harmed for capitals which are not multiples of ticket prices. However, one can see that the envelope of maxima of this curve is perfectly egalitarian. The spikes that cause the discontinuity of the curve are due to the large ticket price (currently \$1756), which in Decred is determined by the market and is high due to the limited supply of tickets available per ticket pool, a parameter inherent in their protocol. Perfect egalitarianism could in principle be achieved by making the ticket price approach 0.

Table 2 A comparison of the egalitarianism values of the cryptocurrencies explored in this study.

Name	Consensus mechanism	Egalitarianism
Bitcoin	Proof-of-work	-0.034490298
Ethereum	Proof-of-work	-0.006926114
Litecoin	Proof-of-work	-0.000271822
Monero	Proof-of-work	-0.002206135
Decred	Proof-of-work Proof-of-stake	-0.412524642 -0.000348280
Ouroboros	Proof-of-stake	-0.000000295

In the case of Ouroboros, every coin has the same probability of being chosen for eligibility [19]. When a coin is eligible for block generation, its owner can create a block by providing a proof of ownership of the chosen coin. Consider the case of a cryptocurrency with N coins in circulation. When a block needs to be created, a coin is chosen at random from the set of N coins. Therefore, each coin may be chosen with $\frac{1}{N}$ probability. Then the address which owns the chosen coin, in other words the stakeholder which controls this coin, is eligible to generate a block and receive the block rewards associated with it[6]. In our experiments, we assume that every block is associated with a constant reward, which pertains to newly minted coins. Furthermore, since computational power does not affect the rate of block production, it is reasonable to assume that both the electricity and the hardware equipment's price is constant for all users, regardless of stake accumulation, so all users can participate using – relatively – cheap resources (cf. Remark 1).

Figure 2b depicts the simulation of a pure proof-of-stake system. In this case, the users pay a set transaction fee for the purchase of the initial stake. The rest of their capital is allocated as stake. The figure suggests that this system is closer to perfect egalitarianism compared to the rest of our case studies.

Summary. Our findings are summarized in Table 2. We find that Bitcoin is the least egalitarian, followed in turn by Ethereum, Monero, and Litecoin[7]. The latter two are the most egalitarian due to their use of CryptoNight and scrypt respectively. Mining with Decred provides the worst egalitarianism of all tested coins. However, the most egalitarian coins involve staking. Decred staking, due to its quantized ticket pricing, is only approximately egalitarian and comparable to the performance of mining Litecoin. Pure proof-of-stake, which allows continuous staking, is *almost perfectly* egalitarian, its small divergence from perfect egalitarianism stemming from the small capital which is required to pay the transation fees[8] to participate in the staking process.

6 Conclusion

In this work we explore the notion of *egalitarianism* of cryptocurrencies. Although this notion has long been discussed, we are the first to give a definition, which allows us to concretely argue about the egalitarianism of various existing systems.

[6] This slightly deviates from the work of [19] in the fact that the payout in our case is due to freshly minted coins and not fees. However, the reward schedule is identical.

[7] Litecoin may appear to have better egalitarianism compared to Monero due to limited availability of mining machines. More data are needed to economically compare scrypt and CryptoNight mining.

[8] As of January 2019, according to `https://cardanoexplorer.com/`, in the Cardano implementation of Ouroboros these fees are in the order of $0.01.

The results of our experimental simulations are very optimistic in terms of usability of our metric, as they provide concrete figures which measure the egalitarianism of several popular cryptocurrencies. The most exciting result arises from the comparison between the proof-of-work and proof-of-stake mechanisms. Although blockchain folklore argued in favour of proof-of-work systems in terms of egalitarianism, our results show that, in fact, it is proof-of-stake systems which are more egalitarian.

Our work provides the first step towards establishing a concrete framework of egalitarianism evaluation in the cryptocurrency ecosystem. Future work will focus in evaluating more existing cryptocurrencies and investigating variations of consensus mechanisms such as delegated proof-of-stake. Additionally, we leave for future work the treatment of more complex economical models of the mining game such as dynamic systems and adversarial strategies, as well as economies of scale in the multitude of parameters we have ignored, such as electricity bulk pricing. We conjecture the consideration of such parameters will exacerbate the gap between proof-of-work and proof-of-stake which we have illustrated in this work.

References

1 U.S. Energy Information Administration. Electric Power Monthly with Data for November 2018. Technical report, U.S. Energy Information Administration, January 2019. URL: https://www.eia.gov/electricity/monthly/current_month/epm.pdf.

2 Joël Alwen, Jeremiah Blocki, and Krzysztof Pietrzak. Depth-robust graphs and their cumulative memory complexity. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 3–32. Springer, 2017.

3 Alex Biryukov and Dmitry Khovratovich. Egalitarian Computing. In *USENIX Security Symposium*, pages 315–326, 2016.

4 Joseph Bonneau, Andrew Miller, Jeremy Clark, Arvind Narayanan, Joshua A. Kroll, and Edward W. Felten. SoK: Research perspectives and challenges for bitcoin and cryptocurrencies. In *2015 IEEE Symposium on Security and Privacy*, pages 104–121. IEEE Computer Society Press, May 2015. doi:10.1109/SP.2015.14.

5 Lars Brünjes, Aggelos Kiayias, Elias Koutsoupias, and Aikaterini-Panagiota Stouka. Reward Sharing Schemes for Stake Pools. Computer Science and Game Theory (cs.GT) arXiv:1807.11218, 2018. arXiv:1807.11218.

6 Vitalik Buterin et al. A next-generation smart contract and decentralized application platform. *white paper*, 2014.

7 Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget. *arXiv preprint*, 2017. arXiv:1710.09437.

8 dEBRYUNE and dnaleor. PoW change and key reuse. Available at: https://www.getmonero.org/2018/02/11/PoW-change-and-key-reuse.html, February 2018. URL: https://www.getmonero.org/2018/02/11/PoW-change-and-key-reuse.html.

9 The Decred Developers. Decred Documentation. Available at: https://docs.decred.org/, 2016. URL: https://docs.decred.org/.

10 John R Douceur. The sybil attack. In *International workshop on peer-to-peer systems*, pages 251–260. Springer, 2002.

11 Cynthia Dwork and Moni Naor. Pricing via Processing or Combatting Junk Mail. In Ernest F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 139–147. Springer, Heidelberg, August 1993.

12 Ittay Eyal and Emin Gün Sirer. Majority Is Not Enough: Bitcoin Mining Is Vulnerable. In Nicolas Christin and Reihaneh Safavi-Naini, editors, *FC 2014*, volume 8437 of *LNCS*, pages 436–454. Springer, Heidelberg, March 2014. doi:10.1007/978-3-662-45472-5_28.

13 Giulia Fanti, Leonid Kogan, Sewoong Oh, Kathleen Ruan, Pramod Viswanath, and Gerui Wang. Compounding of Wealth in Proof-of-Stake Cryptocurrencies. In *International Conference on Financial Cryptography and Data Security*. Springer, 2019.

14 Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The Bitcoin Backbone Protocol: Analysis and Applications. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 281–310. Springer, Heidelberg, April 2015. doi:10.1007/978-3-662-46803-6_10.

15 Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The Bitcoin Backbone Protocol with Chains of Variable Difficulty. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 291–323. Springer, Heidelberg, August 2017.

16 Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

17 Olga Kharif. Many Bitcoin Miners Are at Risk of Turning Unprofitable. *Bloomberg*, April 2018. URL: https://www.bloomberg.com/news/articles/2018-04-18/bitcoin-miners-facing-a-shakeout-as-profitability-becomes-harder.

18 Aggelos Kiayias, Elias Koutsoupias, Maria Kyropoulou, and Yiannis Tselekounis. Blockchain mining games. In *Proceedings of the 2016 ACM Conference on Economics and Computation*, pages 365–382. ACM, 2016.

19 Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 357–388. Springer, Heidelberg, August 2017.

20 Daniel Larimer and the EOS developers. EOS.IO Technical White Paper v2, 2017. URL: https://github.com/EOSIO/Documentation/commits/master/TechnicalWhitePaper.md.

21 Charles Lee. Litecoin, 2011.

22 George B Mathews. On the partition of numbers. *Proceedings of the London Mathematical Society*, 1(1):486–490, 1896.

23 Robert McMillan. Ex-Googler Gives the World a Better Bitcoin. *WIRED*, August 2013. URL: https://www.wired.com/2013/08/litecoin/.

24 Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Available at: https://bitcoin.org/bitcoin.pdf, 2008. URL: https://bitcoin.org/bitcoin.pdf.

25 Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 643–673. Springer, 2017.

26 Rafael Pass and Elaine Shi. FruitChains: A fair blockchain. In Elad Michael Schiller and Alexander A. Schwarzmann, editors, *36th ACM PODC*, pages 315–324. ACM, July 2017.

27 Colin Percival and Simon Josefsson. The scrypt password-based key derivation function. Technical report, Internet Engineering Task Force, 2016.

28 Ayelet Sapirshtein, Yonatan Sompolinsky, and Aviv Zohar. Optimal Selfish Mining Strategies in Bitcoin. In Jens Grossklags and Bart Preneel, editors, *FC 2016*, volume 9603 of *LNCS*, pages 515–532. Springer, Heidelberg, February 2016.

29 Michael Bedford Taylor. Bitcoin and the age of bespoke silicon. In *Proceedings of the 2013 International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, page 16. IEEE Press, 2013.

30 Nicolas Van Saberhagen. CryptoNote v2.0, 2013. URL: https://cryptonote.org/whitepaper.pdf.

31 Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151:1–32, 2014.

A Proofs

▶ **Lemma 3** (Sybil strategies). *Fix a cryptocurrency c and an investment period interval d. Given capital v, for every natural number $i \in \mathbb{N}^\star$, it holds that $f_{c,d}(v) \leq f_{c,d}(i \cdot v)$.*

Proof. We prove the statement via contradiction. Assume that for capital v exists a natural number $i \in \mathbb{N}^\star$ such that $f_{c,d}(v) > f_{c,d}(i \cdot v)$. Also assume that for capital v the optimal strategy is B', so: $\max_{B \in \mathbb{B}} \mathbb{E}[B(v)] = \mathbb{E}[B'(v)]$. Then, for capital $i \cdot v$ exists a strategy B'', such that the capital is split into i equally-sized parts and the strategy B' is applied on each part. Given that the executions of the substrategies on these parts are independent, then the expected returns for the strategy B'' are:

$$\mathbb{E}[B''(i \cdot v)] = i \cdot \mathbb{E}[B'(v)] = i \cdot \max_{B \in \mathbb{B}} \mathbb{E}[B(v)] \tag{1}$$

It also holds that B'' is at best the optimal strategy, so:

$$\max_{B \in \mathbb{B}} \mathbb{E}[B(i \cdot v)] \geq \mathbb{E}[B''(i \cdot v)] \overset{(1)}{\Longrightarrow} \max_{B \in \mathbb{B}} \mathbb{E}[B(i \cdot v)] \geq i \cdot \max_{B \in \mathbb{B}} \mathbb{E}[B(v)] \tag{2}$$

However, it should hold that:

$$\begin{gathered} f_{c,d}(v) > f_{c,d}(i \cdot v) \Rightarrow \\ \frac{\max_{B \in \mathbb{B}} \mathbb{E}[B(v)] - v}{v} > \frac{\max_{B \in \mathbb{B}} \mathbb{E}[B(i \cdot v)] - i \cdot v}{i \cdot v} \overset{(2)}{\Longrightarrow} \\ \frac{\max_{B \in \mathbb{B}} \mathbb{E}[B(v)] - v}{v} > \frac{i \cdot \max_{B \in \mathbb{B}} \mathbb{E}[B(v)] - i \cdot v}{i \cdot v} \Rightarrow \\ \frac{\max_{B \in \mathbb{B}} \mathbb{E}[B(v)] - v}{v} > \frac{\max_{B \in \mathbb{B}} \mathbb{E}[B(v)] - v}{v} \end{gathered} \tag{3}$$

which is impossible. ◀

B Integer programming formulation

In our experiments, we used a Dynamic Programming solution to solve the Knapsack problem in order to allocate mining machines upfront. An optimal solution could use the proceeds of mining not only to reinvest in electricity, but also to purchase new machines. This is captured by the Integer Programming formulation in Figure 3, which gives the optimal investment strategy in the full model.

This maximization problem tries to optimize the *freshly* generated proceeds. The variables to solve for, $x_{m,t} \in \mathbb{N}$, describe the number of machines of type m that the investor holds at time t. We assume machines cannot be sold back to the market, hence $x_{m,t-0} \leq x_{m,t}$. The investment starts with initial capital v and no machines, hence $x_{m,0} = 0$. The program can then decide to purchase machines as time goes by. For any costs, it first uses up the initial capital v to pay for them (this initial capital is useless to keep, as it does not contribute to freshly generated proceeds, which are our utility here), and subsequently uses the proceeds to pay for any remaining costs. Capital which is not expended to pay for costs is discarded by the max operator in the maximization clause. The condition the integer program is subject to requires that the investment has non-negative capital at every point in time, and hence does not run out of money. In this formulation, it is assumed that d is a set of consecutive integers representing indexed *hours* of execution (a more fine-grained solution can be obtained by increasing this temporal resolution as needed).

Maximize

$$\max(0, v - \sum_{t \in d \setminus d[0]} \sum_{m \in \overline{m}} (x_{m,t} - x_{m,t-1}) \text{IC}(m_i) - \sum_{t \in d} \sum_{m \in \overline{m}} x_{m,t} \cdot \text{ECR}(m) \cdot \text{EC})$$

$$+ \sum_{t \in d} \sum_{m \in \overline{m}} x_{m,t} \cdot 3600 \cdot \frac{\text{HR}(m)}{\text{THR}(c)} \cdot \text{BR}(c) \cdot \text{BGR}(c) \cdot \text{TP}(c)$$

subject to

$$\sum_{\substack{t' \leq t \\ t' \neq d[0]}} \sum_{m \in \overline{m}} (x_{m,t'} - x_{m,t'-1})(-\text{IC}(m_i) + (t - t' + 1)|\mathbb{E}[\text{IR}(m)]) \leq v \text{ for } t \in d$$

$$x_{m,t-1} \leq x_{m,t} \text{ for } m \in \overline{m} \text{ and } t \in d \setminus d[0]$$

$$x_{m,d[0]} = 0 \text{ for } m \in \overline{m}$$

and $x_{m,t} \in \mathbb{N}$ for $m \in \overline{m}$ and $t \in d$

Figure 3 An Integer Programming formulation of the optimal investment strategy in our model.

C Parameters affecting egalitarianism

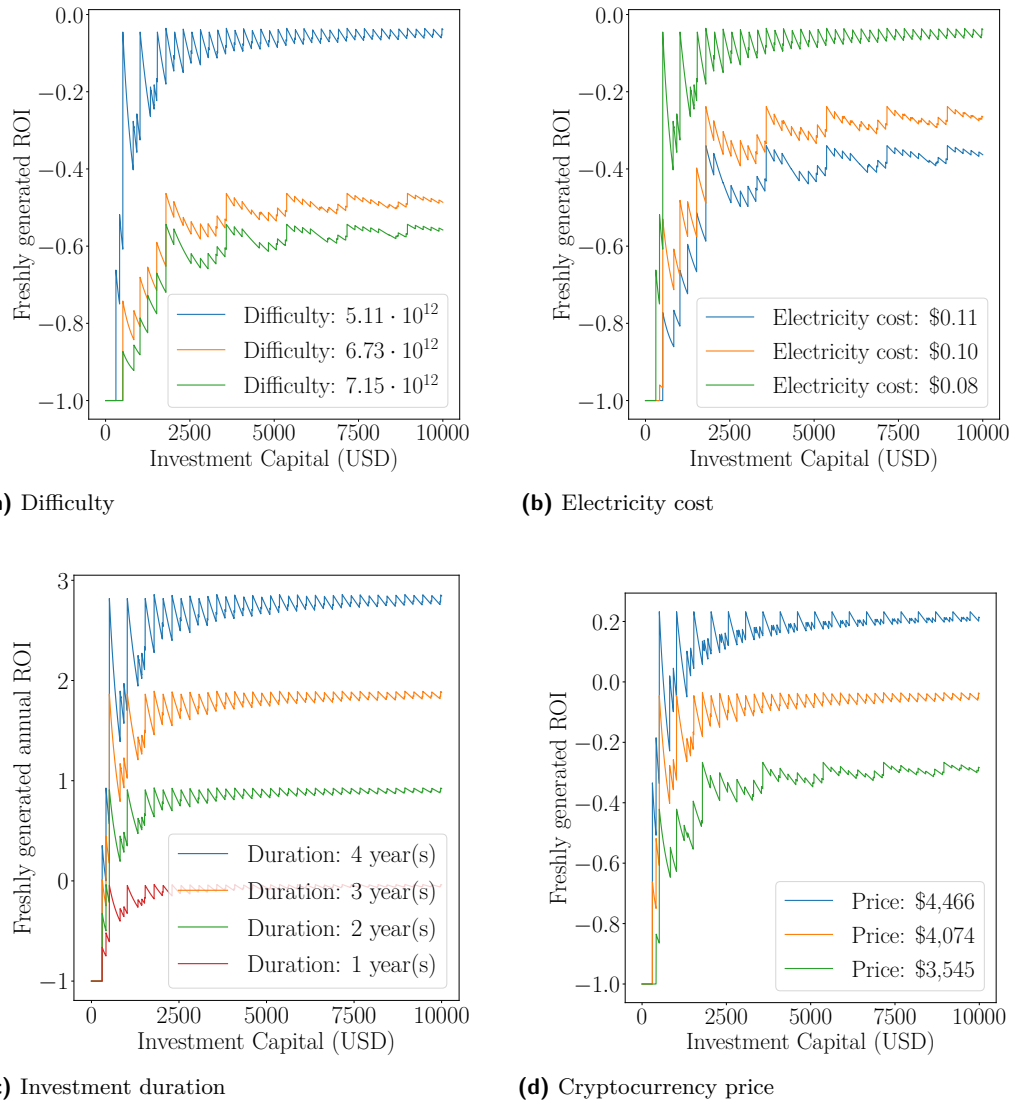
Throughout this paper, we have assumed certain parameters (cryptocurrency prices, electricity prices, duration of investment and mining difficulty) remain constant throughout the investment period. Furthermore, we have taken into account current market values to the best of our knowledge. We note that, while the actual egalitarianism numbers may change depending on these parameters, the general shape of egalitarian curves and the qualitative comparison between different cryptocurrencies remains the same. To illustrate this point, we have measured the egalitarian curve of Bitcoin for varying parameter values. Our results are demonstrated in Figure 4.

D Machines

Data for mining machines was obtained from a multitude of resources on the Internet[9]. Data for graphics processing units (GPU) and central processing units (CPU) was obtained by calculating the median of multiple user benchmarks when available[10]. The price of each machine used in our experiments is the reported retails price of machine at date of access. When a new machine is not available for sale, the price of a used or refurbished machine is

[9] An exhaustive list of our resources includes the online stores `https://whattomine.com/`, `https://cryptominer.deals/`, `https://www.asicminervalue.com/`, `https://www.reddit.com/r/MoneroMining/comments/9omjfb/rtx_2080_ti_mining_monero_at_1228hs_and_more/`, `https://www.newegg.com/`, `https://www.amazon.com/`, `https://shop.bitmain.com.cn`, `https://www.cryptouniverse.at`, `https://canaan.io`, `http://miner.ebang.com.cn`, `https://swminershop.com`, `https://asicminer.co`, `https://estrahash.com`, `http://www.innosilicon.com`, `https://pangolinminer.com`, `https://www.bitfily.io`, `https://hashdeploy.net/`, `https://www.pantech.company`, `https://www.cryptominerbros.com`, `https://pandaminer.com`, `https://minersdeals.com`, `https://sharkmining.com`, `https://shop.miningstore.com`, `https://mineshop.eu`, `https://www.bitmart.co.za`, `https://shop.futurebit.io`, `https://www.aliexpress.com`, `https://bitech-mining.com`, `https://asicminermarket.com`, `https://www.baikalminer.com`, `https://prominerz.com`

[10] `https://www.xmrstak.com/tag/monero/`, `https://gpustats.com/`, `https://www.ethmonitoring.com/benchmark`, `https://monerobenchmarks.info/`



(a) Difficulty

(b) Electricity cost

(c) Investment duration

(d) Cryptocurrency price

Figure 4 Bitcoin egalitarian curves under varying parameters.

used. For reproducibility purposes, our complete data set is openly available in our repository. For reference, we list a summary of those machines which provide a *positive net gain per hour* after purchase (and can thus be profitable under our assumed parameter values) in Table 3.

Table 3 Machines used in experiments.

Bitcoin			
Name	Hashes / s	Watt	Price (USD)
8 Nano Pro	$76 \cdot 10^{12}$	4,000	6,000
Whatsminer M10S	$55 \cdot 10^{12}$	3,500	2,558
Ebit E11++	$44 \cdot 10^{12}$	1,980	2,024
8 Nano	$44 \cdot 10^{12}$	2,100	1,790
T3 43T	$43 \cdot 10^{12}$	2,100	2,279
Ebit E11+ 37	$37 \cdot 10^{12}$	2,035	1,517
WX6	$34 \cdot 10^{12}$	3,200	1,275
Whatsminer M10	$33 \cdot 10^{12}$	2,145	1,022
T2T-32T	$32 \cdot 10^{12}$	2,200	1,568
Ebit E11	$30 \cdot 10^{12}$	1,950	1,110
Antminer S15 (28T)	$28 \cdot 10^{12}$	1,596	1,249
Antminer S15 (27T)	$27 \cdot 10^{12}$	1,539	1,363
T2T-25T	$25 \cdot 10^{12}$	2,050	1,150
Snow Panther B1+	$24.5 \cdot 10^{12}$	2,100	580
T2T-24T	$24 \cdot 10^{12}$	1,980	1,350
S11i	$24 \cdot 10^{12}$	2,300	937
Antminer T15	$23 \cdot 10^{12}$	1,541	840
Antminer S11	$20.5 \cdot 10^{12}$	1,435	512
AvalonMiner 921	$20 \cdot 10^{12}$	1,800	415
Antminer S9-Hydro	$18 \cdot 10^{12}$	1,728	713
Ebit E10	$18 \cdot 10^{12}$	1,650	2,999
T2 Terminator	$17.2 \cdot 10^{12}$	1,570	1,118
DragonMint T1	$16 \cdot 10^{12}$	1,480	1,600
AvalonMiner 851	$15 \cdot 10^{12}$	1,450	380
Antminer S9i	$14.5 \cdot 10^{12}$	1,365	440
Antminer S9j	$14.5 \cdot 10^{12}$	1,365	307
AvalonMiner 841	$13.6 \cdot 10^{12}$	1,290	354.44
SX6i	$11 \cdot 10^{12}$	900	419
Ethereum			
Name	Hashes / s	Watt	Price (USD)
A10 EthMaster	$485 \cdot 10^{6}$	850	5,399
A10 EthMaster	$432 \cdot 10^{6}$	740	4,799
Shark Extreme 2 (8×NVIDIA GTX 1080 Ti)	$420 \cdot 10^{6}$	1,500	9,779
Maximus+ (8×1080TI)	$370 \cdot 10^{6}$	2,200	7,520
A10 EthMaster	$365 \cdot 10^{6}$	650	4,099
Ethereum Mining Rig (12x AMD RX 570 GPU)	$360 \cdot 10^{6}$	1,600	4,345
ULTRON (8×P104)	$320 \cdot 10^{6}$	1,700	5,338
Ethereum Mining Rig (8× NVIDIA 1080 8GB GPU)	$310 \cdot 10^{6}$	1,100	6,267
Shark Extreme 2 (6×NVIDIA GTX 1080 Ti)	$300 \cdot 10^{6}$	1,200	7,880
Shark Extreme 2 (8×AMD Vega 56)	$290 \cdot 10^{6}$	1,700	6,879

Shark Extreme 2 (8×NVIDIA GTX 1070 Ti 8 GB)	$245 \cdot 10^6$	1,400	6,679
Shark Extreme 2 (8×AMD RX 580)	$240 \cdot 10^6$	1,100	4,590
Ethereum Mining Rig (8×AMD MSI RX 580 GPU)	$240 \cdot 10^6$	1,000	3,453
IMPERIUM+ (8×RX 570/580)	$230 \cdot 10^6$	1,300	3,577
Antminer G2	$220 \cdot 10^6$	1,200	3,799
Shark Extreme 2 (6×AMD Vega 56)	$220 \cdot 10^6$	1,275	5,680
Ethereum Mining Rig (8×AMD MSI RX 570 GPU)	$220 \cdot 10^6$	950	3,2253
Shark Extreme 2 (4×NVIDIA GTX 1080 Ti)	$210 \cdot 10^6$	800	4,979
Antminer E3	$190 \cdot 10^6$	760	654
Shark Extreme 2 (6×NVIDIA GTX 1070 Ti 8 GB)	$185 \cdot 10^6$	1,050	5,480
Shark Extreme 2 (6×AMD RX 580)	$180 \cdot 10^6$	825	3,890
Ethereum Mining Rig (6×AMD RX580 8gb GPU)	$180 \cdot 10^6$	900	2,342
Ethereum Mining Rig (6×AMD MSI RX 580 GPU)	$175 \cdot 10^6$	860	1,967
Ethereum Mining Rig (6×AMD MSI RX 580 GPU)	$170 \cdot 10^6$	750	2,156
Thorium 6580 GPU	$160.2 \cdot 10^6$	700	4,297
Thorium 6570 GPU	$144 \cdot 10^6$	750	3,974
Shark Extreme 2 (4×NVIDIA GTX 1070 Ti 8 GB)	$122 \cdot 10^6$	600	3,580
Zodiac 6-1060 GPU	$120.78 \cdot 10^6$	750	3,222
Shark Extreme 2 (4×AMD RX 580)	$120 \cdot 10^6$	550	2,590
Ethereum Mining Rig (6×AMD MSI RX 560)	$80 \cdot 10^6$	370	1,823
GeForce RTX 2080Ti	$55 \cdot 10^6$	155	1,249
GeForce GTX 1080Ti	$51.11 \cdot 10^6$	175	999
RX Vega 64	$44 \cdot 10^6$	230	399
GeForce RTX 2080	$41 \cdot 10^6$	105	699
GeForce GTX TITAN X	$40 \cdot 10^6$	250	1,099
P104-100	$38.89 \cdot 10^6$	127	569
RX Vega 56	$38.75 \cdot 10^6$	210	339
GeForce RTX 2070	$38.5 \cdot 10^6$	140	499
GeForce GTX 1080	$34.07 \cdot 10^6$	121	633
RX 580	$31.3 \cdot 10^6$	110	185
GeForce GTX 1070	$31.1 \cdot 10^6$	108	319
GeForce GTX 1070Ti	$30.83 \cdot 10^6$	107	489
RX 570	$29.85 \cdot 10^6$	65	142
RX 480	$29.71 \cdot 10^6$	70	237
RX 470	$29 \cdot 10^6$	60	340
GeForce GTX 1060 (6GB)	$23.81 \cdot 10^6$	95	264
GeForce GTX 1060 (3GB)	$19.32 \cdot 10^6$	69	189
GeForce GTX 1050Ti	$13.18 \cdot 10^6$	75	169
Litecoin			
Name	Hashes / s	Watt	Price (USD)

A6 LTCMaster	$123 \cdot 10^7$	1,500	3,000
A4+ LTCMaster	$62 \cdot 10^7$	750	1,500
Apollo LTC Pod	$10 \cdot 10^7$	100	299

Monero			
Name	Hashes / s	Watt	Price (USD)
Shark Extreme 2 (8×AMD Vega 56)	14,800	1,700	6,879
Shark Extreme 2 (6×AMD Vega 56)	11,000	1,275	5,680
Shark Extreme 2 (8×AMD RX 580)	6,880	1,100	4,590
Shark Extreme 2 (6×AMD RX 580)	5,160	825	3,890
Shark Extreme 2 (4×AMD RX 580)	3,440	550	2,590
RX Vega 64	2,020	140	399
RX Vega 56	1,920	140	339
GeForce RTX 2080Ti	1,200	150	1,249
RX 580	976	89	185
RX 480	965	140	237
Ryzen Threadripper 1920X	955	140	435
GeForce RTX 2080	898	132	699
GeForce GTX 2070	880	140	499
RX 470	840	120	340
GeForce GTX 1070	777	112	319
RX 570	740	90	142
Ryzen 7 2700X	715	105	309
Ryzen 5 1600X	532	47	179
Ryzen 5 1600	531	65	159

Decred			
Name	Hashes / s	Watt	Price (USD)
Whatsminer D1	$44 \cdot 10^{12}$	2,200	1,588
Whatsminer DCR	$44 \cdot 10^{12}$	2,200	1,890
Antminer DR5	$35 \cdot 10^{12}$	1,610	1,282
STU-U1+	$12.8 \cdot 10^{12}$	1,850	1,560
STU-U1	$11 \cdot 10^{12}$	1,600	1,389

The Stability and the Security of the Tangle

Quentin Bramas
ICUBE, University of Strasbourg, CNRS, France
bramas@unistra.fr

Abstract

In this paper we study the stability and the security of the distributed data structure at the base of the IOTA protocol, called the Tangle. The contribution of this paper is twofold. First, we present a simple model to analyze the Tangle and give the first discrete time formal analyzes of the average number of unconfirmed transactions and the average confirmation time of a transaction.

Then, we define the notion of *assiduous honest majority* that captures the fact that the honest nodes have more hashing power than the adversarial nodes *and* that all this hashing power is constantly used to create transactions. This notion is important because we prove that it is a necessary assumption to protect the Tangle against double-spending attacks, and this is true for any tip selection algorithm (which is a fundamental building block of the protocol) that verifies some reasonable assumptions. In particular, the same is true with the Markov Chain Monte Carlo selection tip algorithm currently used in the IOTA protocol.

Our work shows that either all the honest nodes must constantly use all their hashing power to validate the main chain (similarly to the Bitcoin protocol) or some kind of authority must be provided to avoid this kind of attack (like in the current version of the IOTA where a coordinator is used).

The work presented here constitute a theoretical analysis and cannot be used to attack the current IOTA implementation. The goal of this paper is to present a formalization of the protocol and, as a starting point, to prove that some assumptions are necessary in order to defend the system again double-spending attacks. We hope that it will be used to improve the current protocol with a more formal approach.

2012 ACM Subject Classification Networks → Network performance modeling; Networks → Network security

Keywords and phrases Distributed Ledger Technology, Security, Stability

Digital Object Identifier 10.4230/OASIcs.Tokenomics.2019.8

Related Version A previous version of the paper is available at `https://hal.archives-ouvertes.fr/hal-01716111`.

1 Introduction

Since the day Satoshi Nakamoto presented the Bitcoin protocol in 2008 [5], the interest in Blockchain technologies has grown continuously. More generally, this interest concerns *Distributed Ledger Technology*, which refers to a distributed data storage protocol. Usually it involves a number of nodes (or processes, or agents) in a network that are known to each other or not. Those nodes may not trust each-other so the protocol should ensure that they reach a consensus on the order of the operations they perform, in addition to other mechanisms like data replication for instance.

The consensus problem has been studied for a long time [1, 6] providing a number of fundamental results. But the solvability of the problem was usually given in terms of proportion of faulty agents over honest agents. In a trustless network, where anyone can participate, an adversary can simulate an arbitrary number of nodes in the network. To avoid that, proof systems like Proof of Work (PoW) or Proof of Stake (PoS) are used to link the importance of an entity with some external properties (processing power in PoW) or internal properties (the number of owned tokens in PoS) instead of simply the number

of nodes it controls. The consensus problem is now solvable only if the importance of the adversary (given in terms of hashing power or in stake) is smaller than the honest one (the proportion is reduced to 1/3 if the network is asynchronous).

In Bitcoin and in the other blockchain technologies, transactions are stored in a chain of blocks, and the PoW or PoS is used to elect one node that is responsible for writing data in the next block. The "random" selection and the incentive for a node to execute honestly the protocol make the whole system secure, as it was shown by several formal analysis [2, 5]. Usually, there are properties that hold with high probability i.e., with a probability that tends to one quickly as the time increases. For instance, the order between two transactions does not change with probability that tends to 1 exponentially fast over the time in the Bitcoin protocol, if the nodes executing honestly (or rationally) the protocol have more than a third of the hashing total power.

In this paper we study another distributed ledger protocol called *the Tangle*, presented by Serguei Popov [7], that is used in the IOTA cryptocurrency to store transactions. The Tangle is a Directed Acyclic Graph (DAG) where a vertex, representing a transaction, has two parents, representing the transactions it confirms.

According to the protocol a PoW Puzzle must be solved to add a transaction to the Tangle. This PoW prevents an adversary from spamming the network. However, it is not clear in the definition of the Tangle how this PoW impacts its security.

When a new transaction is appended to the Tangle, it references two previous unconfirmed transactions, called tips. The algorithm selecting the two tips is called a *Tip Selection Algorithm* (TSA). It is a fundamental parts of the protocol as it is used by the participants to decide, among two conflicting transactions, which one is valid. It is the most important part in order for the participants to reach a consensus. The TSA currently used in the IOTA implementation uses the PoW contained in each transaction to select the two tips.

Related Work

Very few academic papers exist on this protocol, and there is no previous work that formally analyzes its security. The white paper behind the Tangle [7] presents a quick analysis of the average number of transactions in the continuous time setting. This analysis is done after assuming that the number of tips converge toward a stationary distribution. The same paper presents a TSA using *Monte Carlo Markov Chain* (MCMC) random walks in the DAG from old transactions toward new ones, to select two unconfirmed transactions. The random walk is weighted to favor transactions that are confirmed by more transactions. There is no analysis on how the assigned weight, based on the PoW of each transaction affects the security of the protocol. This MCMC TSA is currently used by the IOTA cryptocurrency.

It is shown in [8] that choosing the default TSA is a Nash equilibrium. Participants are encouraged to use the MCMC TSA, because using another TSA (e.g. a lazy one that confirms only already confirmed transactions) may increase the chances of seeing their transactions unconfirmed.

Finally, the tangle has also been analyzed by simulation [3] using a discrete time model, where transactions are issued every round following a Poisson distribution of parameter λ. Like in the continuous time model, the average number of unconfirmed transactions (called *tips*) seems to grow linearly with the value of λ, but a little bit slower ($\approx 1.26\lambda$ compared to 2λ in the continuous time setting).

Contributions

The contribution of our paper is twofold. First, we analyze formally the number of tips in the discrete time setting, depending on the value of λ by seeing it as a Markov chain where at each round, there is a given probability to obtain a given number of tips. Unlike previous work, we here prove the convergence of the system toward a stationary distribution. This allows us to prove the previous results found by simulations [3] that the average number of tips is stationary and converge towards a fixed value.

Second, we prove that if the TSA depends only on the PoW, then the weight of the honest transactions should exceed the hashing power of the adversary to prevent a double-spending attack. This means that honest nodes should constantly use their hashing power and issue new transactions, otherwise an adversary can attack the protocol even with a small fraction of the total hashing power. Our result is interesting because it is true for any tip selection algorithm i.e., the protocol cannot be more secure by simply using a more complex TSA.

The remaining of the paper is organized as follow. Section 2 presents our model and the Tangle. In Section 3 we analyze the average confirmation time and the average number of unconfirmed transactions. In Section 4 we prove our main theorem by presenting a simple double-spending attack.

2 Model

2.1 The Network

We consider a set $\mathcal{N}$ of processes, called nodes, that are fully connected. Each node can send a message to all the other nodes (the network topology is a complete graph).

We assume nodes are activated synchronously. The time is discrete and at each time instant, called round, a node reads the messages sent by the other nodes in the previous round, executes the protocol and, if needed, broadcast a message to all the other nodes. When a node broadcasts a message, all the other nodes receive it in the next round. Those assumptions are deliberately strong as they make an attack more difficult to perform, which is useful when studying necessary assumptions.

2.2 The DAG

In this paper, we consider a particular kind of distributed ledger called *the Tangle*, which is a Direct Acyclic Graph (DAG). Each node u stores at a given round r a local DAG G_r^u (or simply G_r or G if the node or the round are clear from the context), where each vertex, called *site*, represents a transaction. Each site has two parents (possibly the same) in the DAG. We say a site *directly confirms* its two parents. All sites that are confirmed by the parents of another site are also said to be confirmed (or indirectly confirmed) by it i.e., there is a path from a site to all the sites it confirms in the DAG (see Figure 1). A site that is not yet confirmed is called a *tip*. There is a unique site called *genesis* that does not have parents and is confirmed by all the other sites. For simplicity we identify a DAG simply by a set of sites $G = (s_i)_{i \in I}$

Two sites may be *conflicting*. This definition is application-dependent so we assume that there exists a function $areConflicting(a, b)$ that answer whether two sites are conflicting or not.

If the Tangle is used to store the balance of a given currency (like the IOTA cryptocurrency), then a site represents a transaction moving funds from a sender address to a receiver address and two sites are conflicting if they try to move the same funds to two different receivers i.e., if both executing transactions results in a negative balance for the sender. The

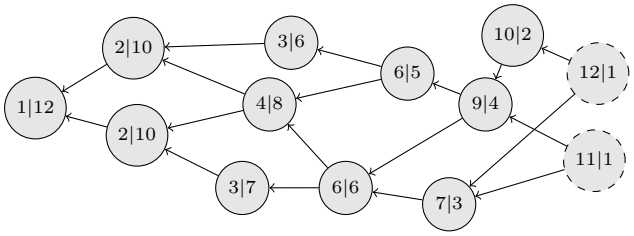


Figure 1 An example of a Tangle where each site has a weight of 1. In each site, the first number is its score and the second is its cumulative weight. The two tips (with dashed border) are not confirmed yet and have cumulative weight of 1.

details of this example are outside the scope of this paper, but we may use this terminology in the remaining of the paper. In this case, signing a transaction means generating a site, and broadcasting a transaction means sending it to the other nodes so that they can include it to their local Tangle.

At each round, each node may sign one or more transactions. For each transaction, the node selects two parents. The signed transaction becomes a site in the DAG. Then, the node broadcasts the site to all the other nodes.

DAG extension

▶ **Definition 1.** *Let G be a DAG and A a set of sites, disjoint from G. If each site of A has its parents in A or in the tips of G, then we say that A is an extension of G and $G \cup A$ denotes the DAG composed by the union of sites from G and A. We also say that A extends G.*

One can observe that if A extends G, then the tips of G form a cut of $G \cup A$.

▶ **Definition 2.** *Let A be a set of sites extending a DAG G. We say A completely extends G (or A is a complete extension of G) if all the tips of $G \cup A$ are in A. In other word, the sites of A confirm all the tips of G.*

▶ **Definition 3.** *A DAG of a node may contain conflicting sites. If so, the DAG is said to be forked (or conflicting). A conflict-free sub-DAG is a sub-DAG that contains no conflicting sites.*

Weight and Hashing Power

When a transaction is signed, a small proof of work (PoW) has to be solved to include it in the DAG. The difficulty of this PoW is called the weight of the site. Initially, this PoW has been added to the protocol to prevent a node from spamming a huge number of transactions. In order to issue a site of weight w a processing power (or *hashing power*) proportional to w needs to be consumed.

With the PoW, spamming requires a large amount of processing power, which increases its cost and reduces its utility. It was shown [7] that sites should have bounded weight and for simplicity, one can assume that the weight of each site is 1.

Then, this notion is also used to compute the *cumulative weight* of a site, which is the amount of work that has been done to deploy this site and all sites that confirm it. Similarly, *the score* of a site is the sum of all weight of sites confirmed by it i.e., the amount of work that has been done to generate the sub-DAG confirmed by it (see Figure 1 for an illustration).

2.3 Tip Selection Algorithm

When signing a transaction s, a node u has to select two parents i.e., two previous sites in its own version of the DAG. According to the protocol, this is done by executing an algorithm called the *tip selection algorithm* (TSA). The protocol says that the choice of the parents must be done among the sites that have not been confirmed yet i.e., among tips. Also, the two selected parents must not confirm, either directly or indirectly, conflicting sites.

We denote by $\mathcal{T}$ the TSA, which can depend on the implementation of the protocol. For simplicity, we assume all the honest nodes use the same algorithm $\mathcal{T}$. As pointed by previous work [7], the TSA is a fundamental factor of the security and the stability of the Tangle.

For our analysis, we assume $\mathcal{T}$ depends only on the topology of the DAG, on the weight of each site in the DAG and on a random source. With those assumptions, we say the TSA is *stateless*. If it also depends on previous output of the TSA it is said to be *stateful*.

The output of $\mathcal{T}$ depends on the current version of the DAG and on a random source (that is assumed distinct for two different nodes). The random source is used to prevent different nodes that has the same view from selecting the same parents when adding a site to the DAG at the same time. However, this is not deterministic and there are not guaranties i.e., it is possible that two distinct nodes issue two sites with the same parents.

Local Main DAG

The local DAG of a node u may contain conflicting sites. For consistency, a node u can keep track of a conflict-free sub-DAG $main_u(G)$ that it considers to be its local *main* DAG. If there are two conflicting sites a and $\bar{a}$ in the DAG G, the local main DAG contains at most one of them.

The main DAG of a node is used as a reference for its own view of the world, for instance to calculate the balance associated with each address. Of course, this view may change over the time. When new transactions are issued, a node can change its main DAG, updating its view accordingly (exactly like in the Bitcoin protocol, when a fork is resolved due to new blocks being mined). When changing its main DAG, a local node may discard a sub-DAG in favor of another sub-DAG. In this case, several sites may be discarded. This is something we want to avoid or at least ensure that the probability for a site to be discarded tends quickly to zero with time.

The tip selection algorithm decides what are the tips to confirm when adding a new site. Implicitly, this means that the TSA decides what sub-DAG is the main DAG. In more detail, the main DAG of a node at round r is the sub-DAG confirmed by the two sites output by the TSA. Thus, a node can run the TSA just to know what its main DAG is, even if no site has to be included to the DAG.

One can observe that, to reach consensus, the TSA should ensure that the main DAG of all the nodes contain a common prefix of increasing size that represents the transactions everyone agree on.

2.4 Adversary Model

Among the nodes, some are honest i.e., they follow the protocol, and some are byzantine and behave arbitrarily. For simplicity, we can assume that only one node is byzantine and we call this node *the adversary*. The adversary is connected to the network and receive all the transactions like any other honest node. He can behave according to the protocol but he can also create (and sign) transactions without broadcasting them, called hidden transaction (or hidden sites).

When an honest node issues a new site, the two sites the $\mathcal{T}$ outputs must be two tips, at least in the local DAG of the node. Thus, one parent p_1 cannot confirm indirectly the other p_2, because in this case the node is aware of p_2 having a child and not being a tip. Also, a node cannot select the same site as parent for two different site, thus the number of honest children cannot exceed the number of nodes in the network. This implies the following property.

▶ **Property 1.** *In a DAG constructed by n honest nodes using a TSA, a site cannot have one parent that confirms the other one. Moreover, the number of children of each site is bounded by the number n of nodes in the network.*

The first property should be preserved by an adversary as it is easy for the honest nodes to check and discard a site that does not verify it. However the adversary can issue multiple sites that directly confirm the same site and the honest nodes have no way to know which sites are honest.

Assiduous Honest Majority Assumption

The cumulative weight and the score can be used by a node to select its main DAG. However, even if it is true that a heavy sub-DAG is harder to generate than a light one, there is no relation yet in the protocol between the weight of sites and the hashing power capacity of honest nodes.

We define the *assiduous honest majority assumption* as the fact that the hashing power of honest nodes is constantly used to generate sites and that it is strictly greater than the hashing power of the adversary. In fact, without this assumption, it is not relevant to look at the hashing power of the honest nodes if they do not constantly use it to generates new sites.

Thus, under this assumption, the cumulative weight of the honest DAG grows according to the hashing power of the honest nodes, and the probability that an adversary generates more sites than the honest nodes in a given period of time tends to 0 as the duration of the period tends to infinity. Conversely, without this assumption, an adversary may be able to generates more sites than the honest nodes, even with less available hashing power.

3 Average Number of Tips and Confirmation Time

In this section we study the average number of tips depending on the rate of arrival of new sites. In this section, like in previous analysis [7], we assume that tip selection algorithm is the simple uniform random tip selection that selects two tips uniformly at random.

We denote by $N(t)$ the number of tips at time t and $\lambda(t)$ the number of sites issued at time t. Like previously, we assume $\lambda(t)$ follows a Poisson distribution of parameter λ. Each new site confirms two tips and we denote by $C(t)$ the number of tips confirmed at time t. We have:

$$N(t) = N(t-1) + \lambda(t) - C(t).$$

We say we are in state $N \leq 1$ if there are N tips at time t. Then, the number of tips at each round is a Markov chain $(N(t))_{t\geq 0}$ with an infinite number of states $[1, \infty)$. To find the probability of transition between two states (given in Lemma 5) we first calculate the probability of transition when the number of new site is known.

▶ **Lemma 4.** *If the number of tips is N and k new sites are issued, then the probability $P_{N \xrightarrow{k} N'}$ of having N' tips in the next round is:*

$$P_{N \xrightarrow{k} N'} = \frac{N!}{N^{2k}(N'-k)!} \left\{ {2k \atop N-N'+k} \right\}$$

where $\left\{ {a \atop b} \right\}$ denotes the Stirling number of the second kind $S(a,b)$.

Proof. If k new sites are issued, then there are up to $2k$ sites that are confirmed. This can be seen as a "balls into bins" problem [4] with $2k$ balls thrown into N bins, and the goal is to see how many bins are not empty i.e. how many unique sites are confirmed.

First, there are N^{2k} possible outcome for this experience so the probability of a particular configuration is $\frac{1}{N^{2k}}$. The number of ways we can obtain exactly $C = N - N' + k$ non empty bins, or confirmed transaction (so that there are exactly N' tips afterward) is the number of ways we can partition a set of $2k$ elements into C parts times the number of ways we can select C bins, in a given order, to receive those C parts (also known as the C-permutations of $2k$).

The first number is called the Stirling number of the second kind and is denoted by $\left\{ {2k \atop N-N'+k} \right\}$. The second number is $\frac{N!}{(N'-k)!}$. ◀

Then, the probability of transition is a direct consequence of the previous lemma

▶ **Lemma 5.** *The probability of transition from N to N' is*

$$P_{N \to N'} = \sum_{k=|N'-N|}^{N'} \mathbb{P}(\Lambda = k) P_{N \xrightarrow{k} N'} = \sum_{k=|N'-N|}^{N'} \frac{N! \lambda^k e^{-\lambda}}{N^{2k}(N'-k)!k!} \left\{ {2k \atop N-N'+k} \right\}.$$

Proof. We just have to observe that the probability of transition from N to N' is null if the number of new sites is smaller than $N - N'$ (because each new site can decrease the number of tips by at most one), smaller than $N' - N$ (because each site can increase the number of tips by at most one), or greater than N' (because each new site is a tip). ◀

▶ **Lemma 6.** *The Markov chain $(N(t))_{t \geq 0}$ has a positive stationary distribution π.*

Proof. First, it is clear that $(N(t))_{t \geq 0}$ is aperiodic and irreducible because for any state $N > 0$, resp. $N > 1$, there is a non-null probability to move to state $N + 1$, resp. to state $N - 1$. Since it is irreducible, we only have to find one state that is positive recurrent (i.e., that the expectation of the hitting time is finite) to prove that there is a unique positive stationary state.

For that we can observe that the probability to transition from state N to $N' > N$ tends to 0 when N tends to infinity. Indeed, for a fixed k, we even have that the probability to decrease the number of tips by k tends to 1:

$$P_{N \xrightarrow{k} N-k} = \frac{N!}{N^{2k}(N-2k)!} = (1 - \frac{1}{N})(1 - \frac{2}{N}) \dots (1 - \frac{2k+1}{N}) \tag{1}$$

$$\lim_{N \to \infty} P_{N \xrightarrow{k} N-k} = 1 \tag{2}$$

so that for any $\varepsilon > 0$ there exists a k_ε such that $\mathbb{P}(\Lambda \geq k_\varepsilon) < \varepsilon/2$ and from (2) an N_ε such that $\forall k < k_\varepsilon$, $P_{N \xrightarrow{k} N-k} < \frac{\varepsilon}{2k_\varepsilon}$. So we obtain:

$$A_{N_\varepsilon} = \sum_{N'>N\geq N_\varepsilon} P_{N\to N'} = \mathbb{P}(N(i+1) > N(i+1) \mid N(i) \geq N) \tag{3}$$

$$< \mathbb{P}(\Lambda \geq k_\varepsilon) + \sum_{k<k_\varepsilon} P_{N \overset{k}{\to} N-k} \tag{4}$$

$$< \varepsilon \tag{5}$$

so that the probability A_N to "move away" from states $[1, N]$ tends to 0 when N tends to infinity. In fact, it is sufficient to observe that there is a number $N_{1/2}$ such that the probability to "move away" from states $[1, N_{1/2}]$ is strictly smaller than $1/2$. Indeed, this is a sufficient condition to have a state in $[1, N_{1/2}]$ that is positive recurrent (one can see this by looking at a simple random walk in one dimension with a mirror at 0 and a probability $p < 1/2$ to move away from 0 by one and $(1-p)$ to move closer to 0 by 1). From the irreducibility of $(N(t))_{t\geq 0}$, all the states are positive recurrent and the Markov chain admit a unique stationary distribution π. ◀

The stationary distribution

The stationary distribution π verifies the formula $\pi_N = \sum_{i\geq 1} \pi_i P_{i\to N}$, which we can use to approximate it with arbitrary precision. The stationary distribution, for several values of λ is shown in Figure 2.

When the stationary distribution is known, the average number of tips can be calculated $N_{avg} = \sum_{i>0} i\pi_i$, and with it the average confirmation time *Conf* of a tip is simply given by the fact that, at each round, a proportion λ/N_{avg} of tips are confirmed in average. So $Conf = N_{avg}/\lambda$ rounds are expected before a given tip is confirmed. The value of *Conf* depending on λ is shown in Figure 3.

With this, we show that *Conf* converges toward a constant when λ tends to infinity. In fact, for a large λ, the average confirmation time is approximately 1.26, equivalently, the average number of tips $\mathbb{N}_{avg}$ is 1.26λ. For smaller values of λ, intuitively the time before first confirmation diverges to infinity and N_{avg} converges to 1.

When λ tends to infinity

When λ tends to infinity, we can compute the exact average expected confirmation time. When λ is great enough, one can assume the number of tips is well concentrated around its expectation so that we can do the analysis considering only the expected values. Assume that $\lambda \in \mathbb{N}$ sites are being issued and that there are N tips. Each new site confirms uniformly at random 2 tips. If t_n is the average number of tips confirmed after n sites have been chosen among the N previous tips, one can prove the following recursive formula $t_n = t_{n-1}(1 - 1/N) + 1$. Thus, in average the λ new sites confirms

$$t_{2\lambda} = \sum_{i=0}^{2\lambda-1} \left(1 - \frac{1}{N}\right)^i = N\left(1 - \left(1 - \frac{1}{N}\right)^{2\lambda}\right).$$

As we said, in average, every round λ tips should be confirmed, hence $t_{2\lambda} = \lambda$. Which gives us asymptotically when λ (and N) tends to infinity

$$\exp\left(-\frac{2\lambda}{N}\right) = 1 - \frac{\lambda}{N}.$$

By using the Lambert function W, the solution to this equation is

$$\frac{N}{\lambda} = \frac{2}{W(-2e^{-2}) + 2} \approx 1.2550009749159754.$$

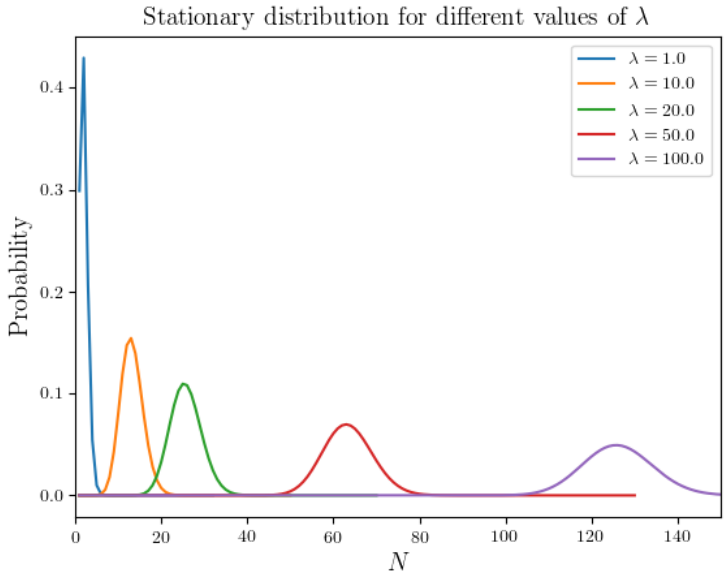


Figure 2 Stationary distribution of the number of tips, for different values of λ. For each value of λ, one can see that the number of tips is really well centered around the average.

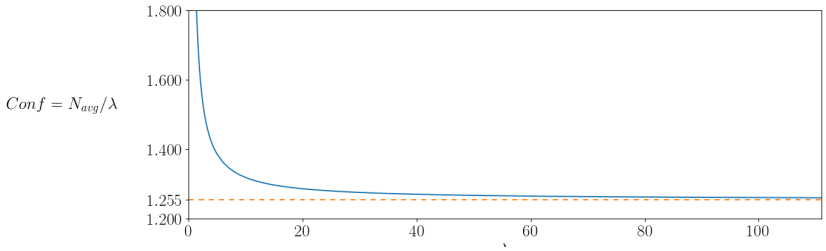


Figure 3 Expected number of round before the first confirmation, depending on the arrival rate of transaction. We see that it tends to 1.26 with λ. Recall that $Conf = N_{avg}/\lambda$ where N_{avg} refers to the average number of tips in the stationary state.

4 A Necessary Condition for the Security of the Tangle

A simple attack in any distributed ledger technology is the double spending attack. The adversary signs and broadcast a transaction to transfer some funds to a seller to buy a good or a service, and when the seller gives the good (he consider that the transaction is finalized), the adversary broadcast a transaction that conflicts the first one and broadcast other new transactions in order to discard the first transaction. When the first transaction is discarded, the seller is not paid anymore and the funds can be reused by the adversary.

The original description of our attack of the Tangle is as follow: after the initial transaction to the seller, the adversary generates the same sites as the honest nodes, forming a sub-DAG with the same topology as the honest sub-DAG (but including the conflicting transaction). Having no way to tell the difference between the honest sub-DAG and the adversarial sub-DAG, the latter will be selected by the honest nodes at some point. This approach may not work with latency in the network, because the sub-DAG of the adversary is always shorter than the honest sub-DAG, which is potentially detected by the honest nodes. To counter this, the adversary can generate a sub-DAG that has not exactly the same topology, but that has the best possible topology for the tip selection algorithm. The adversary can then use all its available hashing power to generate this conflicting sub-DAG that will at some point be selected by the honest nodes.

For this attack we use the fact that a TSA selects two tips that are likely to be selected by the same algorithm thereafter. For simplicity we captured this with a stronger property: the existence of a maximal deterministic TSA.

▶ **Definition 7** (maximal deterministic tip selection algorithm). *A given TSA $\mathcal{T}$ has a maximal deterministic TSA $\mathcal{T}_{det}$ if $\mathcal{T}_{det}$ is a deterministic TSA and for any DAG G, there exists $N_G \in \mathbb{N}$ such that for all $n \in \mathbb{N}$ the following property holds:*

Let A_{det} be the extension of G obtained with $N_G + n$ executions of $\mathcal{T}_{det}$. Let A be an arbitrary extension of G generated with $\mathcal{T}$ of size at most n, conflicting with A_{det}, and let $G' = G \cup A \cup A_{det}$. We have:

$$\mathbb{P}(\mathcal{T}(G') \in A_{det}) \geq 1/2.$$

Intuitively this means that executing the maximal deterministic TSA generates an extension this is more likely to be selected by the honest nodes, provided that it contains more sites than the other extensions. When the assiduous honest majority assumption is not verified, the adversary can use this maximal deterministic TSA at his advantage.

▶ **Theorem 8.** *Without the assiduous honest majority assumption, and if the TSA has a maximal deterministic tip selection, the adversary can discard a transaction that has an arbitrary cumulative weight and achieve double spending.*

Proof. Without the assiduous honest majority assumption, we assume that the adversary can generate strictly more sites than the honest nodes. Let W be an arbitrary weight. One can see W has the necessary cumulative weigh a given site should have in order to be considered final. Let G_0 be the common local main DAG of all node at a given round r_0. At this round our adversary can generate two conflicting sites confirming the same pair of parents. One site a is sent to the honest nodes and the other $\bar{a}$ is kept hidden.

The adversary can use $\mathcal{T}_{det}$ the maximal deterministic TSA of $\mathcal{T}$ to generate continuously (using all its hashing power) sites extending $G \cup \{\bar{a}\}$. While doing so, the honest nodes extend $G \cup \{a\}$ using the standard TSA $\mathcal{T}$. After r_W rounds, it can broadcast all the generated sites to the honest nodes. The adversary can choose r_W so that (i) the probability that it has generated N_G more sites than the honest nodes is sufficiently high, and (ii) transaction a has the target cumulative weight W.

After receiving the adversarial extension, by definition 7, honest nodes will extend the adversarial sub-DAG with probability greater than 1/2. In expectation, half of the honest nodes start to consider the adversarial sub-DAG as their main DAG, thus the honest nodes will naturally converge until they all chose the adversarial sub-DAG as their main DAG, which discard the transaction a.

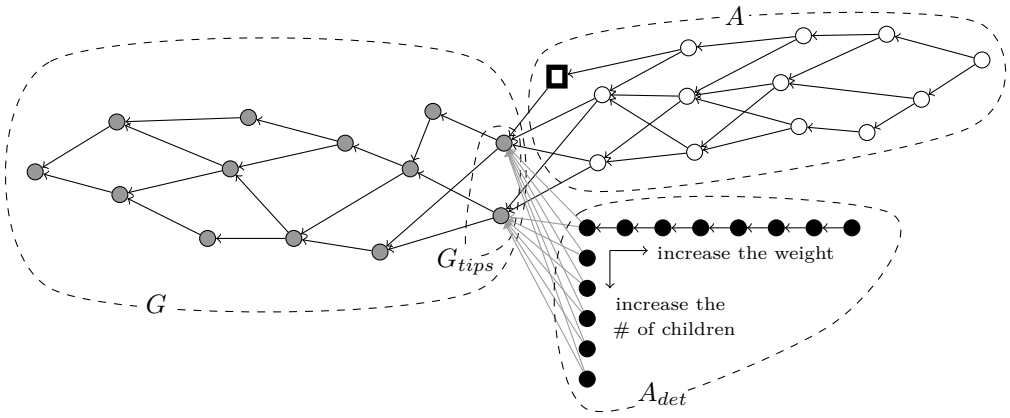


Figure 4 A and A_{det} are two possible extensions of G. The rectangle site conflicts with all site in A_{det} so that when executing the TSA on $G \cup A \cup A_{det}$, tips either from A or from A_{det} are selected. The strategy to construct A_{det} can be either to increase the number of children of G_{tips} or to increase their weight ; both ways are presented here.

If the bandwidth of each channel is limited, then the adversary can start broadcasting the sites of its conflicting sub-DAG at round r_W, at a rate two times greater than the honest nodes. This avoids congestion, and at round $r_W + r_W/2$ all the adversarial sub-DAG is successfully received by the honest nodes. Due to this additional latency, the number of sites in the honest sub-DAG might still be greater than the number of sites in the adversarial sub-DAG, so the adversary continues to generate and to broadcast sites extending its conflicting sub-DAG and at round at most $2r_W$, the adversarial extension of G received by the honest nodes has a higher number of sites than the honest extension.

So the same property is true while avoiding the congestion problem. ◀

Now that we have our main theorem, we show that any TSA defined in previous work (especially in the Tangle white paper [7]) has a corresponding maximal deterministic TSA. To do so we can see that to increase the probability for the adversarial sub-DAG to be selected, the extension of a DAG G obtained by the maximal deterministic TSA should either increase the weight or the number of direct children of the tips of G as shown in Figure 4. We now prove that the three TSA presented in the Tangle white paper [7], (i) the random tip selection, (ii) the MCMC algorithm and (iii) the Logarithmic MCMC algorithm, all have a maximal deterministic TSA, which implies that the assiduous honest majority assumption is necessary when using them (recall that we do not study the sufficiency of this assumption).

4.1 The Uniform Random Tip Selection Algorithm

The uniform random tip selection algorithm is the simplest to implement and the easiest to attack. Since it chooses the two tips uniformly at random, an attacker just have to generate more tips than the honest nodes in order to increase the probability to have one of its tips selected.

▶ **Lemma 9.** *The Random TSA has a maximal deterministic TSA.*

Proof. For a given DAG G the maximal deterministic $\mathcal{T}_{det}$ always chooses as parents one of the l tips of G. So that, after $n + l$ newly added sites A_{det}, the tips of $G \cup A_{det}$ are exactly A_{det} and no other extension of G of size n can produce more than $n + l$ tips so that the probability that the random TSA select a tip from A_{det} is at least $1/2$. ◀

▶ **Corollary 10.** *Without the assiduous honest majority assumption, the Tangle with the Random TSA is susceptible to double-spending attack.*

4.2 The MCMC Algorithm

The MCMC algorithm is more complex than the random TSA. It starts by initially putting a fixed number of walkers on the local DAG. Each walker performs a random walk towards the tips of the DAG with a probabilistic transition function that depends on the cumulative weight of the site it is located to and its children. In more details, a walker at a site v has a probability $p_{v,u}$ to move to a child u with

$$p_{v,u} = \frac{\exp(-\alpha(w(v) - w(u)))}{\sum_{c \in C_v} \exp(-\alpha(w(v) - w(c))} \tag{6}$$

where the set C_v is the children of v, and $\alpha > 0$ is a parameter of the algorithm.

The question to answer in order to find the maximal deterministic TSA of MCMC algorithm is: what is the best way to extend a site v to maximize the probability that the MCMC walker chooses our sites instead of another site. The following Lemma shows that the number of children is an important factor. This number depends on the value α. Indeed the following lemma states that if a site v has constant number C_α of children of weight n, then an MCMC walker at v has a probability at least $1/2$ to move to one of those children, even if we add n other sites to the tangle.

▶ **Lemma 11.** *There exists a constant C_α such that, if an MCMC walker is at an arbitrary site v that has C_α children of weight n, then, when extending v with an arbitrary set of sites H of size n, the probability that the walker moves to H is at most $1/2$.*

Proof. When extending v with n sites, one can choose the number h of direct children, and then how the other sites extends those children. There are several ways to extends those children which changes their weights $w_1, w_2, \ldots, w_h$. The probability p_H for a MCMC walker to move to H is calculated in the following way:

$$\begin{aligned} S_H &= \sum_1^h exp(-\alpha(W - w_i)) \\ \overline{S_H} &= C_\alpha exp(-\alpha(W - n)) \\ S &= S_H + \overline{S_H} \\ p_H &= S_H/S. \end{aligned}$$

The greater the weight the greater the probability p_H. Adding more children, might reduce their weights (since H contains only n sites). For a given number of children h, there are several way to extends those children, but we can arrange them so that each weight is at least $n - h + 1$ by forming a chain of length $n - h$ and by connecting the children to the chain with a perfect binary tree. The height l_i of a children i gives it more weight. So that we have $w_i = n - h + l_i$. A property of a perfect binary tree is that $\sum_1^h 2_i^l = 1$. We will show there is a constant C_α such that for any h and any $l_1, \ldots, l_h$, with $\sum_1^h 2_i^l = 1$, we have

$$\begin{aligned}\overline{S_H} &\geq S_H \\ C_\alpha \exp(-\alpha(W-n)) &\geq \sum_{i=1}^{h} \exp(-\alpha(W-w_i)) \\ C_\alpha &\geq \sum_{i=1}^{h} \exp(-\alpha(h-l_i)). \end{aligned} \tag{7}$$

Surprisingly, one can observe that our inequality does not depend on n, so that the same is true when we arrange the sites when extending a site v in order to increase the probability for the walker to select our sites.

Let $f_h : (l_1, \ldots, l_h) \mapsto e^{-\alpha h} \sum_1^h \exp(\alpha l_i)$. So the goal is to find an upper bound for the function f_h that depends only on α.

The function f_h is convex (as a sum of convex functions), so the maximum is on the boundary of the domain, which is either

$$(l_1, \ldots, l_h) = (1, 2, \ldots, h)$$

or

$$(l_1, \ldots, l_h) = (\lceil \log(h) \rceil, \ldots, \lceil \log(h) \rceil, \lfloor \log(h) \rfloor, \ldots, \lfloor \log(h) \rfloor).$$

For simplicity, let assume that h is a power of two so that the second case is just $\forall i, l_i = \log(h)$.

In the first case we have

$$f_h(1, \ldots, h) = \exp(-\alpha h) \frac{\exp(\alpha(h+1)) - \exp(-\alpha)}{\exp(-\alpha) - 1} = \frac{\exp(\alpha) - \exp(-\alpha(h+1))}{\exp(-\alpha) - 1}$$

which tends to 0 when h tends infinity, so it admits a maximum C_α^1.

In the second case, we have

$$f_h(1, \ldots, h) = \exp(-\alpha h) h \exp(\alpha \log(h))$$

which again tends to 0 when h tends infinity, so it admits a maximum C_α^2 By choosing $C_\alpha = \max(C_\alpha^1, C_\alpha^2)$ we have the inequality (7) for any value of h. ◀

▶ **Lemma 12.** *The MCMC tip selection has a maximal deterministic TSA.*

Proof. Let G be a conflict-free DAG with tips G_{tips}.

Let T be the number of tips times the number C_α defined in Lemma 11. The T first executions of $\mathcal{T}_{det}$ select a site from G_{tips} (as both parents) until each site from G_{tips} has exactly C_α children.

The next executions of $\mathcal{T}_{det}$ selects two arbitrary tips (different if possible). After T executions, only one tip remains and the newly added sites form a chain.

Let $N_G = 2T$. N_G is a constant that depends only on α and on G. After $N_G + n$ added sites, each site in G_{tips} has a C_α children with weight at least n. Thus, by Lemma 11, a MCMC walker located at a site $v \in G_{tips}$ moves to our extension with probability at least $1/2$. Since this is true for all sites in G_{tips} and G_{tips} is a cut. All MCMC walker will end up in A_{det} with probability at least $1/2$. ◀

One can argue that this is not optimal and we could have improved the construction of the extension to reduce the number of sites, but we are mostly interested here in the existence of such a construction. Indeed, in practice, the probability for a walker to move to our extension would be higher as the honest sub-DAG A is not arbitrary, but generated with the TSA. Our analysis shows that even in the worst configuration, the adversary can still generate an extension with a good probability of being selected.

▶ **Corollary 13.** *Without the assiduous honest majority assumption, the Tangle with the MCMC TSA is susceptible to double-spending attack.*

4.3 The Logarithmic MCMC Algorithm

In the Tangle white paper, it is suggested that the MCMC probabilistic transition function can be defined with the function $h \mapsto h^{-\alpha} = \exp(-\alpha \ln(h))$. In more details, a walker at a site v has a probability $p_{v,u}$ to move to a child u with

$$p_{v,u} = \frac{(w(v) - w(u))^{-\alpha}}{\sum_{c \in C_v} (w(v) - w(c))^{-\alpha}} \tag{8}$$

where the set C_v is the children of v, and $\alpha > 0$ is a parameter of the algorithm. The IOTA implementation currently uses this function with $\alpha = 3$.

With this transition function, the number of children is more important than their weight.

▶ **Lemma 14.** *The logarithmic MCMC tip selection has a maximal deterministic TSA.*

Proof. Let G be a conflict-free DAG with tips G_{tips} and T be the number of tips. We construct $\mathcal{T}_{det}$ in the following way. $\mathcal{T}_{det}$ always selects two sites from G_{tips} in a round-robin manner. After kT executions ($k \in \mathbb{N}$), each site from G_{tips} has $2k$ children.

Let n be the number of sites generated with $\mathcal{T}_{det}$. Let $v \in G_{tips}$. We have that the number of children of v generated by $\mathcal{T}_{det}$ is $C_{det} = 2n/T$ (for simplicity we assume that T divides $2n$). Let A be an arbitrary extension of G of size n and C_v be the number of children of v that are in A.

With $w(v)$ the weight of v and $u = \text{argmax}_{x \in A} w(x)$ the child in A with maximum weight, we have that $w(v) \leq 2n$ and $C_{det} \leq w(v) - w(u)$. Let p be the probability that a walker located at v chooses a site generated by $\mathcal{T}_{det}$. We have

$$p \geq \frac{C_{det}(w(v)-1)^{-\alpha}}{C_v(w(v)-w(u))^{-\alpha} + C_{det}(w(v)-1)^{-\alpha}} = \frac{1}{\frac{C_v(w(v)-w(u))^{-\alpha}}{C_{det}(w(v)-1)^{-\alpha}} + 1}.$$

Then

$$\frac{C_v(w(v)-w(u))^{-\alpha}}{C_{det}(w(v)-1)^{-\alpha}} \leq \frac{C_v(C_{det})^{-\alpha}}{C_{det}(2n)^{-\alpha}} = \frac{C_v T^{\alpha}}{C_{det}} = \frac{C_v}{2nT^{1-\alpha}}.$$

With T a constant and C_v bounded (by Property 1), we have that the last fraction tends to 0, and thus p tends to 1, as n tends to infinity. This is true for each site of G_{tips}, so after a given number of generated site N_G, the probability that a LMCMC walker located at any site of G_{tips} moves to a site generated by by $\mathcal{T}_{det}$ is greater than 1/2. ◀

▶ **Corollary 15.** *Without the assiduous honest majority assumption, the Tangle with the Logarithmic MCMC TSA is susceptible to double-spending attack.*

5 Conclusion

We presented a model to analyze the Tangle and we used it to study the average confirmation time and the average number of unconfirmed transactions over the time.

Then, we defined the notion of assiduous honest majority that captures the fact that the honest nodes have more hashing power than the adversarial nodes *and* that all this hashing power is constantly used to create transactions. We proved that for any tip selection algorithm that has a maximal deterministic tip selection (which is the case for all currently known TSA), the assiduous honest majority assumption is necessary to prevent a double-spending attack on the Tangle.

Our analysis shows that honest nodes cannot stay at rest, and should be continuously signing transactions (even empty ones) to increase the weight of their local main sub-DAG. If not, their available hashing power cannot be used to measure the security of the protocol, like we see for the Bitcoin protocol. Indeed, having a huge number of honest nodes with a very large amount of hashing power cannot prevent an adversary from attacking the Tangle if the honest nodes are not using this hashing power. This conclusion may seem intuitive, but the fact that it is true for all tip selection algorithms (that have a deterministic maximal TSA) is something new that has not been proved before.

References

1 Michael J Fischer, Nancy A Lynch, and Michael S Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(2):374–382, 1985.

2 Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 281–310. Springer, 2015.

3 B. Kuśmierz. The first glance at the simulation of the Tangle: discrete model, 2016. URL: `http://iota.org/simulation_tangle-preview.pdf`.

4 Michael Mitzenmacher and Eli Upfal. *Probability and computing: Randomized algorithms and probabilistic analysis*. Cambridge university press, 2005.

5 Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. URL: `https://bitcoin.org/bitcoin.pdf`.

6 M. Pease, R. Shostak, and L. Lamport. Reaching Agreement in the Presence of Faults. *J. ACM*, 27(2):228–234, April 1980. `doi:10.1145/322186.322188`.

7 S Popov. The tangle. White paper, 2016. URL: `https://iota.org/IOTA_Whitepaper.pdf`.

8 Serguei Popov, Olivia Saa, and Paulo Finardi. Equilibria in the Tangle. *arXiv preprint*, 2017. `arXiv:1712.05385`.

The Impact of Ethereum Throughput and Fees on Transaction Latency During ICOs

Michael Spain
School of Computer Science, University of Sydney, Australia
mspa1382@uni.sydney.edu.au

Sean Foley
School of Business, University of Sydney, Australia
sean.foley@sydney.edu.au

Vincent Gramoli
School of Computer Science, University of Sydney, Australia
Data61-CSIRO, Sydney, Australia
vincent.gramoli@sydney.edu.au

Abstract

The Ethereum blockchain has gained popularity for its ability to implement Initial Coin Offerings (ICOs), whereby a buyer enters a market order agreement with a seller in order to purchase cryptographic tokens at an agreed price. The popularity of ICOs in 2017 has created an increasingly adversarial environment among potential buyers, who compete for what is often a fixed supply of tokens offered for a limited period of time.

We study the impact of a series of ICOs in order to understand the relationship between transaction fees, throughput and latency in Ethereum. Our analysis considers the effects on both Ethereum's service providers, known as miners, and users who issue transactions in the network. Our results show that while buyers incentivise miners generously to include their transactions during ICOs, the latency of these transactions is predominantly determined by the levels of supply and demand in the network.

2012 ACM Subject Classification Security and privacy → Database and storage security; Security and privacy → Economics of security and privacy; Security and privacy → Human and societal aspects of security and privacy

Keywords and phrases ICO, Gas, Ethereum, Transaction Fee, Latency, Fairness

Digital Object Identifier 10.4230/OASIcs.Tokenomics.2019.9

Funding This research is in part supported under Australian Research Council Discovery Projects funding scheme (project number 180104030) entitled "Taipan: A Blockchain with Democratic Consensus and Validated Contracts". Vincent Gramoli is a Future Fellow of the Australian Research Council.

Acknowledgements We wish to thank our colleagues, Ralph Holz and Nate Wiggins, for sharing the Geth Data. We also benefited from the publicly available datasets of Infura and Etherscan.

1 Introduction

An *Initial Coin Offering (ICO)*, which typically consists of offering a fixed quantity of securities at a discounted price for a limited time, has popularized the use of the Ethereum blockchain [21]. Today, Ethereum is the second largest blockchain in terms of market capitalization after Bitcoin [14]. In Ethereum, the notion of gas was introduced in part for the need to incentivise miners to include, in the blocks they create, transactions of varying computational complexity [2]. Transactions may invoke smart contracts that allow a buyer

and seller to transfer tokens at an agreed upon price expressed in Ether (Eth), the native token of Ethereum. This mechanism was used to raise more than $20B throughout 2017 and 2018.[1]

Research has revealed that in Bitcoin, the higher the fee users are willing to pay to the miners for including their transactions, the faster these transactions are included in the blockchain [13]. Interestingly, applying the same strategy in Ethereum could potentially lead to front running, the act of "entering into an equity trade, options or futures contracts with advance knowledge of a block transaction that will influence the price of the underlying security to capitalize on the trade"[2]. A recent study which considered over 3 months worth of Ethereum transactions has shown that most take more than 3 minutes to be included [20], which is long enough to expose pending transactions to the risk of front running by way of issuing a similar transaction with a larger fee. However, it remains unclear whether transaction fees significantly impact the likelihood of successfully purchasing tokens during an ICO.

In this paper, we study this question empirically, first showing that during popular ICOs some participants are paying significantly higher fees. To begin, we retrospectively analyse the revenue and costs of the mining process during the ICO for the Basic Attention Token (BAT), where the entire supply of available tokens was sold in less than 30 seconds for $35M. During this ICO, participants paid on average more than 300× the average fee, providing rewards for miners that were orders of magnitude higher than the additional mining costs incurred during the ICO. These findings support our hypothesis that participants are willing to have their transactions included faster than others in the blockchain to ensure they could purchase tokens. More specifically, we measured the time it takes for transactions to be included in Ethereum's blockchain depending on their associated fees. Although we confirm our hypothesis that the fee of a transaction is inversely correlated to the latency, the correlation is surprisingly low, indicating that during the observed period, transaction fees were not a successful mechanism for front running in Ethereum. This observation, however, does not explain the large discrepancy in transaction latency we observed during this ICO.

To explain this discrepancy, we conducted a thorough analysis of transaction latency on the Ethereum blockchain for a period of 24 days in 2017, which included 19 ICOs, notably TenX and Tezos, that raised altogether more than $700M. By combining our analysis of the fees with the block gas limits in Ethereum, we identified other factors that contribute to the latency of a transaction. In particular, we observed that service demand was greater than the supply, which dramatically raises the latency of some transactions. More precisely, facing the rising popularity in ICOs, the volume of transactions issued exceeded the capability of the service. Then we also observed that, in mid-2017, when the Ethereum gas limit was increased, the capability of the service also increased. To conclude, this capacity analysis revealed that the effect of transaction fees were insignificant due to the high service demand.

The rest of the paper is organised as follows. Section 2 describes the important concepts about ICOs and the Ethereum blockchain. Section 3 illustrates the tremendous increase in Ethereum transaction fees during a successful ICO. Section 4 correlates the increased transaction fees with the latency of transaction. Section 5 indicates how the supply and demand of the service impacts latency. Finally, Section 6 lists the related work and Section 7 concludes.

[1] `https://cointelegraph.com/news/ico-market-2018-vs-2017-trends-capitalization-localization-industries-success-rate`.
[2] `https://www.nasdaq.com/investing/glossary/f/front-running`.

2 Background

A blockchain is a chain of blocks distributed among multiple participating nodes, where *miners* create blocks to include transactions issued by any participating client node [14]. In proof-of-work blockchains, miners solve a computationally intensive cryptographic problem to prove that their block is legitimate.

2.1 Initial coin offering

An Initial Coin Offering (ICO) is a method of raising funds through a blockchain system for mostly blockchain related projects. Ethereum, being the largest blockchain with the ability to conduct ICOs, has experienced hundreds of ICOs in 2017 alone [17]. ICOs are an attractive alternative to other early stage funding processes such as Venture Capital, because they circumvent many of the legal and regulatory requirements and facilitate individuals' participation. Projects are often able to raise significantly more capital through an ICO than is possible with traditional approaches. We focus our study on Ethereum.

2.2 Mining in Ethereum

Miners participating in the Ethereum network run the Ethereum Virtual Machine (EVM) which executes smart contracts. Unlike transactions in Bitcoin, Ethereum transactions can invoke arbitrarily complex functions through smart contracts. This increased functionality requires the protocol to measure the amount of computation each transaction performs for two reasons [2]. First, a miner needs to be able to determine ahead of time whether the transaction they are about to execute will ever finish. Second, there needs to be a mechanism for users to incentivise miners to include computationally intensive transactions. This is why Ethereum uses the concept of *gas*, whose unit represents one computational step in the EVM – all the opcodes in the EVM have a cost measured in gas. Every transaction in Ethereum must include both the gas limit, which is the maximum amount of gas that can be used executing the transaction and the gas price, which is the price, measured in *Wei* ($1\,Eth = 10^{18}\,Wei$), that the sender will pay per unit of gas. If the transaction execution is not finished after the gas limit is reached, the EVM will abort the transaction and revert any state changes. Hence the fee in Ether associated with transaction t in Ethereum is:

$$fee_t = \frac{gas\text{-}price \times gas\text{-}used(t)}{10^{18}}.$$

The Ethereum mining algorithm is Ethash, which is a memory hard algorithm designed to reduce the level of centralisation risks compared to Bitcoin's Hashcash algorithm that is now dominated by centralised pools of Application Specific Integrated Circuits (ASICs).

2.3 Incentives

When a block is created in Ethereum, the miner of the block can vote to increase, decrease or maintain the total gas limit of the next block. This allows the maximum throughput of Ethereum to adjust over time with the capabilities of the miners. The miner of a block b in Ethereum receives 5 Ether plus the sum of the fees for all transactions included in b:

$$reward_b = 5 + \sum_{\forall t \in b} fee_t.$$

Ideally, the miner will include as many transactions as they can (up to the gas limit of the block). However, the block reward usually exceeds the marginal increase in revenue that is gained from including more transactions, since the miner must restart the process each time it includes new transactions (as the block content changes). The primary incentive for the miner is the block reward, rather than the fees gained from filling the block. This becomes a problem when the number of transactions issued starts to approach the maximum theoretical throughput [13].

3 The Basic Attention Token ICO

In this section we study the impact of an ICO that raised $35M in less than 30 seconds on 31st of May 2017 on the Ethereum economy. We show that the Basic Attention Token (BAT) ICO impacted the relationship between mining revenues and costs.

This experiment extends the research of Möser and Böhme [13] to the context of Ethereum, considering the impact of impatient users on mining revenue and costs. We find that high demand for the network could create an inequitable environment for Ethereum users. These findings serve as motivation for our study of transaction latency, presented in Section 4.

3.1 Experimental settings

This experiment studies the transactions confirmed by the Ethereum network during the BAT ICO on the 31st of May 2017, that started with block 3798640 and ended with block 3798642. The data was obtained from the block explorer Etherchain which provides a public API for Ethereum block and transaction data [8]. Data was gathered for a total of 10003 blocks, which includes 5000 before the BAT sale and 5000 after. This number represents roughly one day before and one day after the sale in order to approximate average network conditions, so that the effect of the BAT ICO can be effectively quantified.

3.2 Mining revenue

The analysis compares the average of a variety of metrics in the period directly before and after the BAT ICO with those observed between blocks 3798640 and 3798642.

Table 1 Average statistics vs. BAT blocks.

	Average	Block 3798640	Block 3798641	Block 3798642
Total Block Fee (Eth)	0.08	28.05	35.29	12.14
Total Block Size (Bytes)	6952	10359	9479	5403
Tx (Per Block)	39	85	42	42
Gas Used (Per Block)	2247441	4313308	4262947	4326224

Table 1 compares metrics observed in the three BAT blocks to the cumulative averages for these metrics recorded before and after. We can see that the average transaction fee per block recorded over the period (excluding the BAT ICO) was 0.08 Ether, 314.5× lower than during the ICO.

In Figure 1(left) we plot the total block fees for each block in the dataset. The impact of the BAT ICO on mining revenue is immediately evident. The log scale used on the y-axis is needed to represent an increase of hundreds of times the average fees per block, with a noticeable residual effect in the blocks that immediately followed the ICO. We now show that this level of mining revenue is not proportional to the increased cost incurred by the miners of these blocks or reflected in the performance of the network.

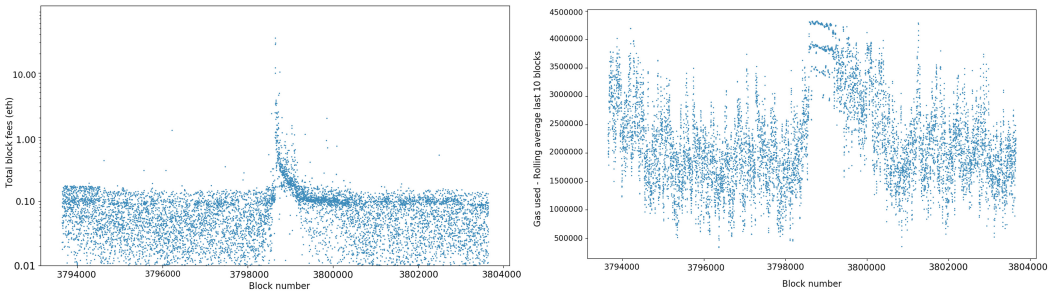


Figure 1 Total fees per block and rolling average of last 10 blocks around the BAT ICO.

The first and second blocks of the BAT sale were 49% and 36% larger than the average block size before and after the ICO, respectively. However, the final BAT block was smaller than the average block size over the period. So, while on aggregate the raw size of the blocks appended by miners during the BAT ICO increased, it was insignificant compared to the increase in revenue. The next metric considered is the number of transactions per block, which reveals the achieved throughput of the network. While the first BAT block recorded twice as many transactions than the average, the other two BAT blocks were close to the average, showing that the network did not provide any significant improvement in throughput during the BAT ICO.

The most appropriate proxy for the computational effort expended by mining is gas used per block. This is because the EVM performs transactions of arbitrary complexity and gas is used to measure the total computational demand of the transaction. The BAT blocks consumed almost twice as much gas as an average block over the period.

Figure 1 (right) depicts a rolling average of gas used in the last 10 blocks. It reveals that around the BAT ICO, the majority of blocks mined were close to the gas limit. This indicates that, although neither block size or throughput increased significantly, the computational work performed by the miners was significantly higher than any other period in the sample.

3.3 Users pay substantial transaction fees during an ICO

Analysing the mining environment around the BAT ICO shows that during periods of high demand in Ethereum, some users are willing to pay massive transaction fees in order to have their transactions included quickly. Consequently, the miners of the BAT blocks received a total block reward that was much larger than usual over the period. In contrast, while the computational costs associated with the mining environment increased, they were insignificant compared to the change in revenue.

This experiment has revealed the actions of impatient users when there is an excessive demand to transact. We hypothesize that due to the fees shown in Figure 1, many users attempting to enter the ICO or transacting during this time were negatively impacted due to miners prioritising high fee transactions first. While in principal it seems fair that high fee transactions should be prioritised, it raises a question of fairness in blockchains. Wealthy users could possibly front run the transactions of others by setting a fee that is large enough.

The cost of immediacy in Ethereum is clearly subject to significant variation based on network activity. This presents a substantial disadvantage to users wishing to transact small during periods of high activity. Whilst it is possible to quantify the effect of large transaction fees on mining revenue and environment, we are not able to determine the effect that these transactions have on other users wishing to transact at a similar time. This observation serves as motivation to study the latency of transactions in Ethereum, to determine how the transaction fee impacts the latency of that transaction.

4 The Impact of Transaction Fees on Latency

As discussed previously, high transaction fees paid by participants during ICOs could possibly be motivated by a front running attempt. In this section, we study how successful high fees are at reducing transaction latency. We start by introducing the concept of transaction latency before describing our experimental setup and conclude that, as expected, transaction fee is inversely correlated to the transaction latency, but surprisingly, that this correlation is negligible.

4.1 Defining transaction latency

We refer to the *latency* of a transaction t as the time taken for it to be included in a block. Note that this does not necessarily correspond to the inclusion time of t, as Ethereum cannot deterministically define the inclusion of a transaction as a number of appended blocks or confirmations [15].

For the latency of transaction t to be well-defined, the block that includes t must be part of the canonical blockchain as determined in Ethereum so that blocks that are part of forks are not considered. Provided that clocks are synchronized, we can say that for some transaction t, $t_{broadcast}$ is the timestamp when the transaction t was initially broadcast, $t_{included}$ is the timestamp of the block creation that included t and $latency_t = t_{included} - t_{broadcast}$, where $t_{included} > t_{broadcast} > 0$.

Unfortunately, we will never know the real value of $t_{broadcast}$ unless the transaction was issued by a node that we controlled. The reason is that the transaction must propagate from the originating node through the peer-to-peer network and the clocks are not perfectly synchronized. We can however approximate latency using the earliest known time for $t_{broadcast}$. Approximating our definition from above, we can say for some transaction t, $t_{received}$ is the earliest timestamp when transaction t is received by some of our nodes, hence:

$$latency_t \approx t_{included} - t_{received}.$$

4.2 Experimental settings

For these experiments, we used two datasets: The first dataset, labelled Geth Data, includes the time at which each transaction was relayed, and the second dataset, labelled Blockchain Data, includes the time at which each transaction was included.

The Geth Data included the timestamps of a transaction when the transaction was relayed in Ethereum using the `geth` client between 2017-06-24 00:00:00 and 2017-07-18 00:00:00. This period includes 19 ICOs: Dao.casino (BET), Pillar (PLR), Mothership (MSP), Blocktix (TIX), TrueFlip (TFL), EOS (EOS), Binance Coin (BNB), InsureX (IXT), CoinDash (CDT), Press.One (PRS), Tezos (XTZ), Nimiq (NET), Polybius (PLBT), Rialto (XRL), Santiment (SAN), Starta (STA), OpenANX (OAX), OmiseGO (OMG), EncryptoTel (ETT) that raised a total of more than $700M [17]. Each time a node received a transaction, it recorded the transaction hash, the $t_{received}$ timestamp, and the IP address of the node that relayed the transaction. This dataset contained 66,472,214 transactions, which is significantly larger than the actual number of transactions included in the blockchain for that period. There are two reasons for this. Firstly, this dataset contained many duplicates of the same transaction as transactions are relayed multiple times in Ethereum. Secondly, this dataset included many transactions that were either never included in a block or part of a fork and not visibly included in the blockchain.

The Blockchain Data were obtained via the blockchain company Infura. They provide a publicly accessible interface to the internal API of a `geth` node. Using this service, we extracted transaction and block data from the Ethereum blockchain for a 24-day period between 24/06/17 00:00:00 - 18/07/17 00:00:00.

In order to determine transaction latency, the datasets were combined by removing duplicate transactions and selecting only the earliest of these timestamps from the Geth Data and matching, using its hash, each transaction in the Blockchain Data to the transaction in the Geth Data.

4.3 The significant variation of latency depending on request time

Figure 2 (left) plots the latency of every transaction committed in the Ethereum blockchain throughout the period observed. Note that the y-axis uses a log scale, indicating that the latency of a transaction varies exponentially depending on the time it was issued. This variation provides a disappointing initial impression of performance. Ideally, latency would not vary by orders of magnitude and should be independent of the time when the transaction was issued. It is noteworthy that Figure 2 (left) does not show the distribution of transaction latency which can give a misleading representation of performance due to outliers.

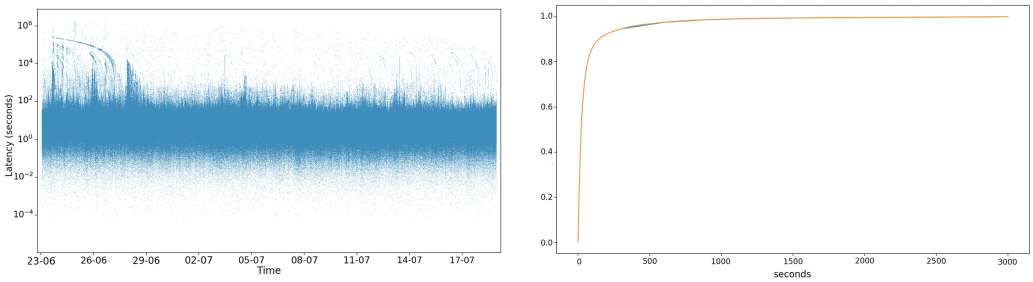


Figure 2 Ethereum transaction latency and empirical cumulative distribution function.

Table 2 shows that the median transaction latency observed was 22.13 seconds. Consider that the average block time for the period was 17.63 seconds. This means that over half of all transactions issued at depth i in the blockchain were included by the block at depth $i + 2$. This observation significantly improves the initial impression given by Figure 2. However, while median performance is strong, the peaks that are seen in Figure 2 (left) are also quantified in the table. The latency data is extremely positively skewed above the 90th percentile.

Table 2 Ethereum transaction latency distribution.

Percentile	10	25	50	75	90	95	99
Latency (seconds)	2.81	8.40	22.13	54.51	158.83	379.22	2854.31

Figure 2 (right) depicts the empirical cumulative distribution function obtained from the empirical study in order to visualise the skew of the latency data. This chart shows there is a point of inflexion in the distribution of latency around the 90th percentile.

The distribution of transaction latency poses an obvious question, why does the shape of the curve changes drastically above the 90^{th} percentile. In other words, what is different about this group of transactions that makes them take significantly longer to be included in a block?

4.4 On the minor impact of transaction fees on latency

The first thing to consider when attempting to explain latency is the transaction fee. All transactions in Ethereum specify a gas price, representing the price the user is willing to pay the miner for each computational step. Since the gas price is at the discretion of the user, perhaps the shape of the data can be explained by the transaction fee. If the gas price is too low, the total transaction fee may not provide enough of an incentive for the miner to include it. In order to examine this hypothesis, we compare the fees paid by transactions that are in the fastest 90% (latency < 158.83 seconds) with the fees paid by transactions in the slowest 10% (latency > 158.83 seconds).

Table 3 Comparison of fee distribution between fastest and slowest transactions.

	25	50	75	90	95	99
Fastest 90% of latency (majority)	5.00^{14}	1.27^{15}	3.90^{15}	8.00^{15}	1.14^{16}	3.15^{16}
Slowest 10% of latency (outliers)	6.60^{14}	1.80^{15}	3.16^{15}	5.10^{15}	8.00^{16}	3.30^{16}

Table 3 shows that the transaction fees paid by the slow outliers are generally higher than in the fast group, except for the 75th and 90th percentiles. This means that these transactions were generally paying a higher fee but experiencing substantially worse latency. Initially, these results seemed counterintuitive. We know that users in a blockchain expect their latency to be correlated with the transaction fee they pay, but these results challenge this assumption. We thus calculated the covariance between the fee and the latency and obtain: $\mathit{covariance}(\mathit{fee}, \mathit{latency}) = -1.453 \times 10^{17}$.

From the covariance we can derive that there is a negative correlation between fee and latency, indicating, as we expect, that the fee is inversely related to the latency:

$$\mathit{correlation}(\mathit{fee}, \mathit{latency}) = -0.0001606.$$

This correlation suggests however that there is very little causal relationship between the transaction fee and latency. Recall that transaction fees in blockchains are supposed to allow a user to incentivise the miner of a block to include a transaction. This statistic challenged the common assumption of users in a blockchain that they can significantly impact the latency of their transaction through the level of the transaction fee.

5 The Impact of Supply and Demand on Latency

The weak correlation between fee and latency raises the question of what is the dominant factor in determining transaction latency. In this section, we study the relation between the supply and demand and how it affects latency. In particular, we study the Ethereum blockchain over the same period as the transactions were issued, where supply increased significantly and deduce the relationship between supply, demand and latency.

5.1 Supply side – gas limit

We are trying to explain why some transactions take significantly longer to be included than others. Our first attempt considering only transaction fees did not only fail to do so, it suggested that the fee itself may be insignificant. In order to try understand these strange results, we now consider the theoretical bounds of the Ethereum network. Recall that in proof-of-work blockchains the only way transactions are included is when a new block is mined:

$$\mathit{capacity\text{-}throughput} = \frac{\mathit{block\text{-}gas\text{-}limit}}{\mathit{block\text{-}interval}}.$$

The maximum number of transactions able to be included in a block is determined by the gas limit, since the Ethereum protocol targets a constant block interval by modifying the difficulty of proof-of-work. In particular, Ethereum's yellow paper [21] states the gas limit H_l of the block must satisfy the following relation: $H_l < P(H)_{H_l} + \frac{P(H)_{H_l}}{1024}$, $H_l > P(H)_{H_l} + \frac{P(H)_{H_l}}{1024}$, where $P(H)_{H_l}$ is the gas limit of the parent block. This mechanism was designed to allow the gas limit to evolve slowly over time to adapt to changes in the mining environment [21]. By allowing each miner to vote independently of one another, Ethereum attempts to avoid some of the centralisation risks in mining by ensuring larger miners are unable to quickly change the gas limit and therefore exclude smaller miners. In effect, Ethereum deliberately makes the gas limit inflexible over shorter periods of time. This means that at any single point in time, Ethereum effectively has a constant maximum throughput. Below we define the maximum number of transactions that can be committed per minute in Ethereum.

We start by taking the median transaction size observed in the study:
Median Transaction Gas = 90,000. We can then approximate the maximum number of transactions per block b in terms of the gas limit and median transaction size:

$$\textit{max-transactions}_b = \textit{block-gas-limit}/90000. \tag{1}$$

The average block interval throughout the study allows us to determine the Average Block Time = 17.63 seconds, and Blocks (Per Minute) = 3.40. Finally, we can derive an approximation for the maximum number of transactions that can be included per minute: $\textit{capacity-throughput} = 3.4 \times \textit{gas-limit}/90000$ transactions/minute.

5.2 Raising the gas limit

Before substituting the gas limit we need, however, to consider a significant event that occurred throughout the study.

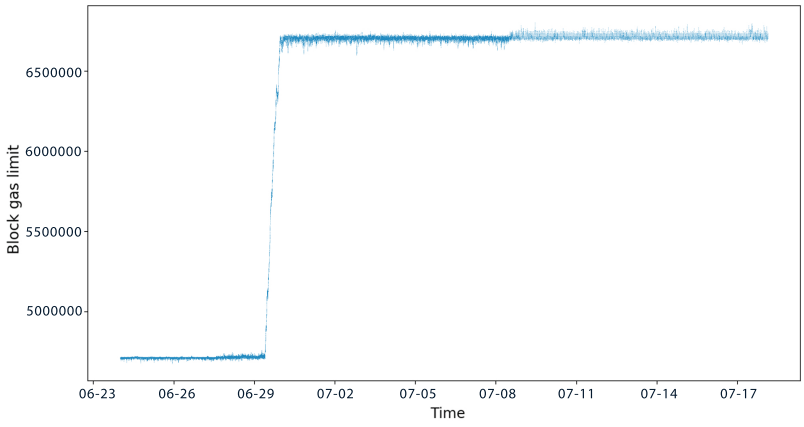


Figure 3 Ethereum block gas limit.

While it was explained that over the short term the gas limit can change only slightly, there was a significant shift observed during the study, shown in Figure 3. On the 29th of June 2017 miners in Ethereum began consistently voting up the gas limit to alleviate

congestion from the increased demand being placed on the network. The result was that the gas limit increased by about 43%. Revising our earlier definition, we can include the average gas limit for each day in the study

$$\textit{block-gas-limit} = \begin{cases} 4711978 & \textit{if Date} < 29\,\textit{June}\,2017, \\ 5348530 & \textit{if Date} = 29\,\textit{June}\,2017, \\ 6711349 & \textit{if Date} > 30\,\textit{June}\,2017. \end{cases}$$

Substituting these block gas limits into Eq. (1) yields the following bounds for the maximum number of transactions that can be committed per minute in Ethereum:

$$\textit{capacity-throughput} = \begin{cases} 178 & \textit{if Date} < 29\,\textit{June}\,2017, \\ 202 & \textit{if Date} = 29\,\textit{June}\,2017, \\ 253 & \textit{if Date} > 30\,\textit{June}\,2017. \end{cases}$$

We now have approximated how many transactions can be included per minute given the relevant gas limit. Essentially these results mean that for the given day, if the number of transactions issued per minute is below the threshold, there should be a strong causal relationship between the transaction fee and the latency. However, it is now necessary to determine the demand placed on the network each minute in order to see the imbalances that occur between demand and supply.

5.3 Demand side – dynamic fluctuations

The Ethereum network has been live since July 30 2015, but there has been a significant increase in popularity of blockchains and cryptocurrency in 2017. This can be seen on the Etherscan website that shows the number of transactions per day, growing from under 50,000 in early 2017 to over 300,000 during the study [9]. Compounding this increased demand has been the hundreds of ICO on Ethereum on the first half of 2017. Throughout the study there were several significant ICOs such as TenX on the 24th of June, which raised 200,000 Ether in around 7 minutes [17].

Figure 4 depicts the number of transactions issued per minute from June to the beginning of July 2017. It appears that there are many minutes where the number of transactions issued significantly exceeds the maximum throughput achievable at that point in time, as discussed in the previous section.

Table 4 Distribution of transactions per minute.

Percentile	25	50	75	90	95	99
Transactions (Per Minute)	130	161	199	244	286	684

Table 4 shows the distribution of this data. Recall that 253 is the maximum possible number of transactions that could be committed per minute after the gas limit raise. This means that almost 10% of the time there were more transactions being issued than could be committed. This statistic is very closely aligned with the empirical cumulative distribution graph of latency in Figure 2 (right), with the shape of that graph changing sharply at the 90th percentile.

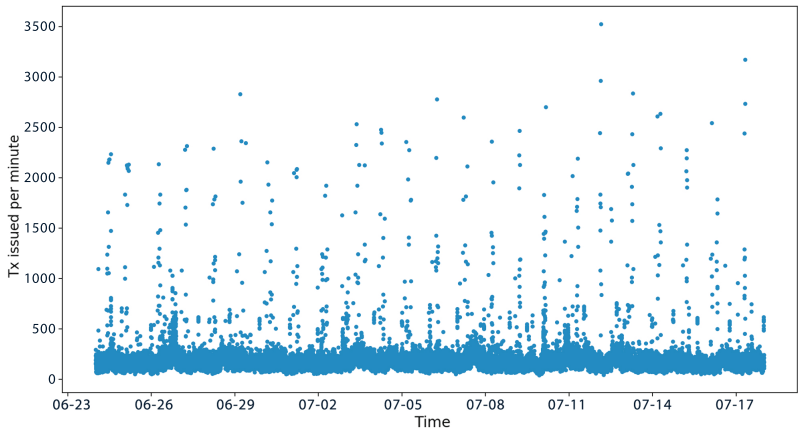


Figure 4 Ethereum transactions issued per minute.

5.4 Median latency per minute

With the understanding of the relationship between demand and supply in Ethereum, we now revisit the latency results. Figure 5 shows the median transaction latency per minute in Ethereum during the period. This graph confirms our initial observation that the Ethereum blockchain typically confirms transactions within approximately 2 blocks. However, as a consequence of rapid fluctuations in demand and a relatively static supply, there are noticeable periods where median latency increases exponentially.

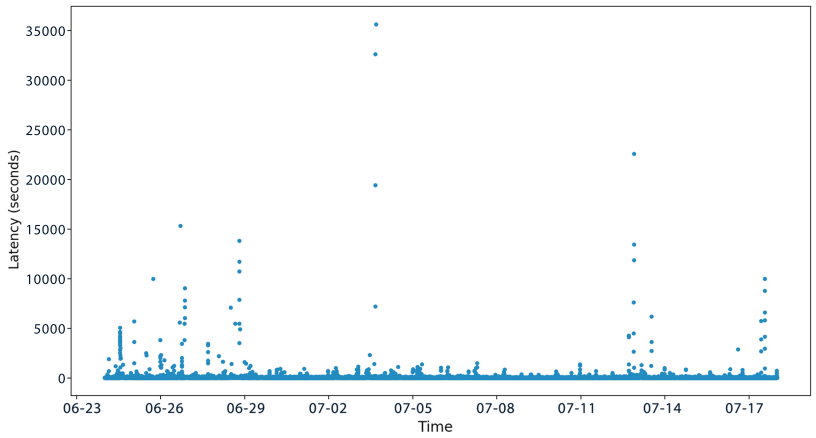


Figure 5 Ethereum transaction median latency per minute.

Table 5 Distribution of median latency per minute.

Percentile	25	50	75	90	95	99
Median Latency (Per Minute)	12.07	19.22	33.88	60.48	87.17	294.47

To recap, we have demonstrated that in Ethereum, the demand is constantly changing and hard to predict, but the supply (how many transactions can be included in the blockchain) is relatively static. This is one of the fundamental performance challenges for Ethereum. The effects of this problem on the users of the blockchain have been relatively insignificant until 2017 where the number of users participating in Ethereum started increasing rapidly.

5.5 Latency vs fee: before and after the gas limit raise

Our work so far has focused on identifying which factors are significant in determining latency in Ethereum. It appears that the transaction fee becomes insignificant in comparison to the overall demand and supply of Ethereum. Fortunately in our study there was a noticeable shift in the gas limit that allows us to analyse the latency vs fee relationship before and after this shift in supply. Figure 6 indicates that the relationship between fees and latency becomes more hyperbolic after the increase, representing the effect of increased Ethereum performance capabilities. The hyperbolic shape more accurately represent the relationship between fee and latency since our data is asymptotic, neither fee nor latency is ever equal to 0.

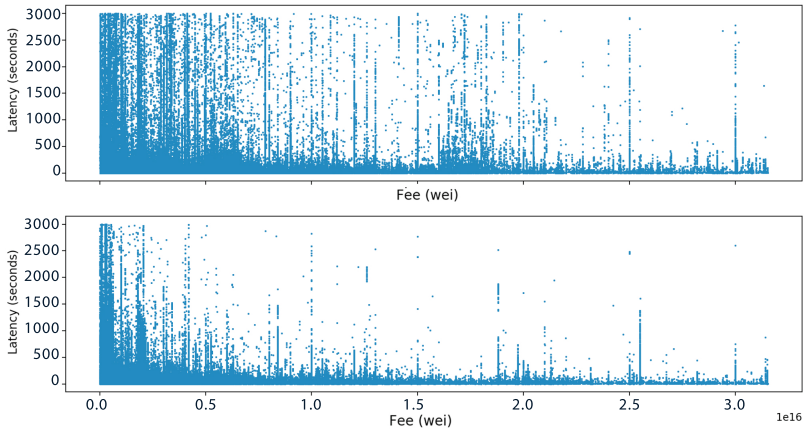


Figure 6 Latency vs fee: before and after the gas limit raise.

6 Related Work

6.1 Proof-of-work and capacity throughput

In a first work [18], Sompolinsky and Zohar analysed the impact that high transaction throughput levels have on the level of security in the Bitcoin protocol. They analysed Bitcoin's longest chain selection method of reaching consensus. Their first contribution was to show that as transaction throughput increases, structural weaknesses in the longest-chain approach makes the network vulnerable to attackers with less computational resources [18].

In a subsequent work, Sompolinsky and Zohar [19] propose the Greedy Heaviest-Observed Sub-Tree (GHOST) consensus algorithm as a means of maintaining the security guarantees while increasing throughput. GHOST trades the longest-chain principle for selecting the heaviest subtree at each fork in the chain. This modification ensures that the work performed by honest nodes is incorporated by the network, even if their blocks do not appear in the final chain. This modification allows the network to deal with the inevitable increase in forks that increased throughput levels cause. They show that while GHOST can scale with the longest-chain algorithm in terms of block creation rate, the primary benefit is a constant security threshold as opposed to exponential decreases in the longest-chain algorithm [19].

6.2 Network and capacity throughput

GHOST helps reduce the security vulnerabilities that forks cause. However, forks still represent a significant weakness in the security of the blockchain as experimented by Natoli and Gramoli [16]. Fundamentally, the underlying network of the blockchain needs to be improved in order to reduce the chance of forks and the vulnerabilities that result from it.

Decker and Wattenhofer [6] analysed information propagation throughout the Bitcoin network in an attempt to determine the primary cause of forks in the blockchain. Their research focuses primarily on identifying improvements in the way the network communicates, by modifying the logical structure of the Bitcoin network. Their motivation is to reduce the number of forks in the blockchain and therefore reduce the decrease in network efficiency that is caused by forks. They identify three significant improvements to the current method of propagation in Bitcoin.

They contend that by dividing the block verification process (that occurs when a node receives a new block) into two components, the initial difficulty check and the validation of transactions, allows for a significant increase in propagation speed. After a node completes the difficulty check and hence verifies the proof-of-work, it can retransmit the block to its peers, before attempting transaction validation. The proposed gain is significant because the majority of work resides in the validation of transactions, whereas verifying proof-of-work is a trivial process [1]. They assert that this modification does not increase the risk of malicious behaviour because producing an invalid block with proof-of-work is just as hard as producing a valid block.

They also suggest that nodes can immediately forward all incoming messages to other nodes, even before actually receiving the block, in an attempt to reduce the round trip time between nodes. While they admit that this does allow an attacker to arbitrarily announce non-existent blocks, attackers are already able to flood the network with fake transactions, and therefore there is no reduction in security. Finally, they suggest that the most significant improvement can be gained by minimising the distance between any two nodes. This can be done by increasing the number of connections that each node maintains, effectively reducing the number of times messages need to be relayed between nodes [6].

6.3 Other approaches to increase the capacity throughput

Decker and Wattenhofer note that the above improvements, while valuable, do little to address what they contend are fundamental structural problems with the network. In a more recent paper, they put forward an entirely different network structure with duplex micropayment channels. They claim that this structure allows vastly superior scalability by deferring to the blockchain for initial setup of a payment channel and conflict resolution, while handling all transactions through the channel itself [7]. Another piece of work by Lewenberg, Sompolinsky and Zohar introduces the inclusion of off-chain blocks into the Bitcoin network. The consequences are similar to that of the GHOST algorithm whereby increased throughput can be achieved, however they also prove that they payoff for weak miners is increased [12].

Kiayias and Panagiotakos [11] also consider the tradeoffs between security and speed, however they extend on the above work by considering multiple blockchains. They introduce a new generic blockchain property, called chain growth, in order to express the minimum rate at which chains of honest parties grow. They derive this property as an extension of their previous work in which they isolate the backbone of the Bitcoin protocol, a useful framework for analysing blockchain fundamentals [10].

The underlying issue we identify here is the security-performance tradeoff that has left blockchains incapable of providing both high security and throughput to its users. Crain et al. [3] recently designed DBFT, a leader-less consensus algorithm to cope with this tradeoff. The algorithm is deterministic and does not assume synchrony, hence guaranteeing that no disagreements can occur, even when the network is behaving badly due to misconfigurations, natural disasters or attacks. The algorithm is also democratic in that it leverages the bandwidth of multiple links rather than relying on the classic leader-based design that is subject to bottlenecks at the leader network interface. The Red Belly Blockchain builds upon this algorithm and an efficient verification sharding protocol to offer a throughput that keeps increasing when increasing the number of consensus participants, typically to hundreds of low-end consensus participant machines [4].

6.4 Transaction fees

Möser and Böhme analyse transaction fees in the Bitcoin blockchain in an attempt to understand the economic and technical components [13]. They examine transactions empirically, in order to determine how fees change over time, and how impatient users incentivise miners to include their transactions. They suggest that the instability of fees over time is a consequence of the protocol failing to provide a mechanism by which users and miners can coordinate to set fair prices. Interestingly, the paper suggests that this issue is not necessarily dangerous as long as mining rewards still dominate the composition of income for miners. This statement raises an interesting question in relation to high throughput which is generally associated with decreasing block rewards [5]. Some information regarding the relation between gas price and confirmation time in Ethereum can be found on the publicly available Eth gas station website[3], however, it does not relate this information to the latency of transactions.

7 Conclusion

In this paper, we analysed the parameters that impact transaction latency in Ethereum. The popularity of ICOs in 2017 has created a competitive environment for users wishing to purchase tokens. While buyers generously incentivised miners to include their transactions, the supply and demand of the service was the predominant factor determining latency and inclusion. For future work, we would like to reproduce the analysis for more recent periods as the Ethereum protocol and network keep evolving.

References

1 T Bamert, C Decker, L Elsen, R Wattenhofer, and S Welten. Have a Snack, Pay with Bitcoins. In *IEEE International Conference on Peer-to-Peer Computing*, 2013.

2 V Buterin. *A Next-Generation Smart Contract and Decentralized Application Platform*, 2013. URL: `https://github.com/ethereum/wiki/wiki/White-Paper`.

3 T Crain, V Gramoli, M Larrea, and M Raynal. DBFT: Efficient leaderless byzantine consensus and its applications to blockchains. In *Proceedings of the 17th IEEE International Symposium on Network Computing and Applications (NCA'18)*, pages 1–8, 2018.

4 T Crain, C Natoli, and V Gramoli. Evaluating the Red Belly Blockchain. Technical report, arXiv, 2018. `arXiv:1812.11747`.

[3] `https://ethgasstation.info/`

5 K Croman, C Decker, I Eyal, A. E Gencer, A Juels, A Kosba, and D Song. On scaling decentralized blockchains. In *International Conference on Financial Cryptography and Data Security*, pages 106–125, 2016.

6 C Decker and R Wattenhofer. Information Propagation in the Bitcoin Network. In *IEEE International Conference on Peer-to-Peer Computing*, pages 1–10, 2013.

7 C Decker and R Wattenhofer. A Fast and Scalable Payment Network with Bitcoin Duplex Micropayment Channels. In *Symposium on Self-Stabilizing Systems*, pages 3–18, 2015.

8 Etherchain. Etherchain Application Programming Interface Block Data, 2017. URL: `https://etherchain.org/documentation/api/`.

9 Etherscan. Etherscan Chart Library, 2017. URL: `https://etherscan.io/charts`.

10 J Garay, A Kiayias, and N Leonardos. The Bitcoin Backbone Protocol: Analysis and Applications. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 281–310, 2015.

11 A Kiayias and G Panagiotakos. Speed-Security Tradeoffs in Blockchain Protocols. *IACR Cryptology ePrint Archive*, 2015.

12 Y Lewenberg, Y Sompolinsky, and A Zohar. Inclusive Blockchain Protocols. In *International Conference on Financial Cryptography and Data Security*, pages 528–547, 2015.

13 M Möser and R Böhme. Trends, Tips, Tolls: A Longitudinal Study of Bitcoin Transaction Fees. In *International Conference on Financial Cryptography and Data Security*, pages 19–33, 2015.

14 S Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*, 2008. URL: `https://bitcoin.org/bitcoin.pdf`.

15 C Natoli and V Gramoli. The Blockchain Anomaly. In *Proceedings of the 15th IEEE Int'l Symp. on Network Computing and Applications*, pages 310–317, 2016.

16 C Natoli and V Gramoli. The Balance Attack or Why Forkable Blockchains Are Ill-Suited for Consortium. In *IEEE/IFIP DSN'17*, June 2017.

17 Smith and Crown. ICOs, token sales, crowdsales, 2017. URL: `https://www.smithandcrown.com/icos/`.

18 Y Sompolinsky and A Zohar. Accelerating Bitcoin's Transaction Processing. Fast Money Grows on Trees, Not Chains. *IACR Cryptology ePrint Archive*, pages 507–527, 2015.

19 Y Sompolinsky and A Zohar. Secure High-Rate Transaction Processing in Bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 507–527, 2015.

20 I Weber, V Gramoli, M Staples, A Ponomarev, R Holz, A B. Tran, and P Rimba. On Availability for Blockchain-Based Systems. In *Proceedings of the 36th International Symposium on Reliable Distributed Systems (SRDS'17)*, pages 64–73. IEEE, September 2017.

21 G Wood. *Ethereum: A Secure Decentralised Generalised Transaction Ledger*, 2014. URL: `http://gavwood.com/paper.pdf`.

F1 Fee Distribution

Dev Ojha
Tendermint, Berkeley, CA, USA
UC Berkeley, CA, USA
dojha@berkeley.edu

Christopher Goes
Tendermint, Berlin, Germany
`https://pluranimity.org/about`
cwgoes@tendermint.com

Abstract

In a proof of stake blockchain, validators need to split the rewards gained from transaction fees each block. Furthermore, these fees must be fairly distributed to each of a validator's constituent delegators. Delegators accrue this reward throughout the entire time which they are delegated, and they have a special operation to withdraw accrued rewards.

The F1 fee distribution scheme works for any algorithm to split fees and inflation between validators each block, with minimal iteration, and the only approximations being due to finite decimal precision. Per block there is a single iteration over the validator set, to enable reward algorithms that differ by validator. No iteration is required to delegate or to withdraw. The state usage is one state update per validator per block and one state entry per active delegation. F1 can optionally handle arbitrary inflation schemes, auto-bonding of rewards, and varying validator commission rates.

2012 ACM Subject Classification Theory of computation → Algorithmic mechanism design

Keywords and phrases Proof of Stake, Fee Distribution, Cosmos

Digital Object Identifier 10.4230/OASIcs.Tokenomics.2019.10

Supplement Material Implementation: `https://github.com/cosmos/cosmos-sdk/tree/develop/x/distribution`

1 Introduction

In a proof of stake blockchain, each validator has an associated stake. Transaction fees are rewarded to validators based on the incentive scheme of the underlying proof of stake model. However, only rewarding the proposers as in many proof-of-work incentive models causes incentive problems. See these prior works discussing this problem. [1] [3] This fee distribution problem occurs in delegated proof-of-stake blockchains, as there is a need to distribute a validator's fee rewards and inflation proportionally to its delegators according to amount of stake each has delegated. The trivial solution of just paying the rewards to each delegator every block is too expensive to perform on-chain, as it would require reading and writing all delegator accounts. Instead fee distribution algorithms must require that delegators perform a withdraw action, which when performed yields the same total amount of fees as if they had received them at every block.

This paper details F1, an approximation-free, slash-tolerant fee distribution algorithm which allows validator commission-rates, inflation rates, and fee proportions, which can all efficiently change per validator, every block. The algorithm requires iterating over the bonded validators every block, which is cheap due to staking logic already requiring iteration over all validators, which causes the expensive state-reads to be cached. Withdraws require no iteration.

International Conference on Blockchain Economics, Security and Protocols (Tokenomics 2019).
Editors: Vincent Danos, Maurice Herlihy, Maria Potop-Butucaru, Julien Prat, and Sara Tucci-Piergiovanni;
Article No. 10; pp. 10:1–10:6

OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The key point of how F1 works is that it tracks how much rewards a delegator with 1 stake delegated to a given validator would be entitled to if it had bonded at block 0 until the latest block. When a delegator bonds at block b, the amount of rewards a delegator with 1 stake would have if bonded at block 0 until block b is also persisted to state. When the delegator withdraws, they receive the difference of these two values. Since rewards are distributed according to stake-weighting, this amount of rewards can be scaled by the amount of stake a delegator had delegated. The following paragraph describes this in more detail, with a demonstration of equivalence to the inefficient iterative algorithm by reduction. Section 2 details how to adapt this algorithm to handle commission rates, slashing, inflation, and auto-bonding of fees.

Base algorithm

In this section, we show that the F1 base algorithm gives each delegator rewards identical to that which they'd receive in the naive and correct fee distribution algorithm that iterated over all delegators every block.

Even distribution of a validators rewards amongst its validators weighted by stake means the following: Suppose a delegator delegates x stake to a validator v at block h. Let the amount of stake the validator has at block i be s_i and the amount of fees they receive at this height be f_i. Then if a delegator contributing x stake decides to withdraw at block n, the rewards they receive are

$$\sum_{i=h}^{n} \frac{x}{s_i} f_i = x \sum_{i=h}^{n} \frac{f_i}{s_i}$$

Note that s_i does not change every block, it only changes if the validator gets slashed, or if any delegator alters the amount they have delegated. We'll relegate handling of slashes to Subsection 2.2, and only consider the case with no slashing here. We can change the iteration from being over every block, to instead being over the set of blocks between two changes in validator v's total stake. Let each of these set of blocks be called a period. A new period begins every time that validator's total stake changes. Let the total amount of stake for the validator in period p be n_p. Let T_p be the total fees that validator v accrued in period p. Let h be the start of period p_{init}, and height n be the end of p_{final}. It follows that

$$x \sum_{i=h}^{n} \frac{f_i}{s_i} = x \sum_{p=p_{init}}^{p_{final}} \frac{T_p}{n_p}$$

Let p_0 represent the period which begins when the validator first bonds. The central idea to the F1 model is that at the end of the kth period, the following is stored at a state location indexable by k: $\sum_{i=0}^{k} \frac{T_i}{n_i}$. Let the index of the current period be f. When a delegator wants to delegate or withdraw their reward, they first create a new entry in state to end the current period. Then this entry is created using the previous entry as follows:

$$Entry_f = \sum_{i=0}^{f} \frac{T_i}{n_i} = \sum_{i=0}^{f-1} \frac{T_i}{n_i} + \frac{T_f}{n_f} = Entry_{f-1} + \frac{T_f}{n_f}$$

Where T_f is the fees the validator has accrued in period f, and n_f is the validators total amount of stake in period f.

The withdrawer's delegation object has the index k for the period which they ended by bonding. (They start receiving rewards for period $k+1$) The reward they should receive when withdrawing is:

$$x \sum_{i=k+1}^{f} \frac{T_i}{n_i} = x \left(\left(\sum_{i=0}^{f} \frac{T_i}{n_i} \right) - \left(\sum_{i=0}^{k} \frac{T_i}{n_i} \right) \right) = x \left(Entry_f - Entry_k \right)$$

It is clear from the equations that this payout mechanism maintains correctness, and requires no iterations. It just needed the two state reads for these entries.

T_f is a separate variable in state for the amount of fees this validator has accrued since the last update to its power. This variable is incremented at every block by however much fees this validator received that block. On the update to the validators power, this variable is used to create the entry in state at f, and is then reset to 0.

This fee distribution proposal is agnostic to how all of the blocks fees are divied up between validators. This creates many nice properties, for example it is possible to only rewarding validators who signed that block.

2 Additional add-ons

2.1 Commission Rates

Commission rates are the idea that a validator can take a fixed $x\%$ cut of all of their received fees, before redistributing evenly to the constituent delegators. This can easily be done as follows:

In block h a validator receives f_h fees. Instead of incrementing that validators "total accrued fees this period variable" by f_h, it is instead incremented by $(1 - commission\_rate) * f_p$. Then $commission\_rate * f_p$ is deposited directly to the validator's account. This allows for efficient updates to a validator's commission rate every block if desired. More generally, each validator could have a function which takes their fees as input, and outputs a set of outputs to pay these fees too. (i.e. x% going to themselves, y% to delegators, z% burnt)

2.2 Slashing

Slashing is distinct from withdrawals, since it lowers the stake of all of the delegator's by a fixed percentage. Since no one is charged gas for slashes, a slash cannot iterate over all delegators. Thus we can no longer just multiply by x over the difference in stake. This section describes a simple solution that should suffice for most chains needs. An asymptotically optimal solution is provided in section 2.4.

The solution here is to instead store each period created by a slash in the validators state. Then when withdrawing, you must iterate over all slashes between when you started and ended. Suppose you delegated at period 0, a y% slash occured at period 2, and your withdrawal creates period 4. Then you receive funds from periods 0 to 2 as normal. The equations for funds you receive for periods 2 to 4 now uses $(1-y)x$ for your stake instead of just x stake. When there are multiple slashes, you just account for the accumulated slash factor.

There is a griefing attack[2] a validator can perform on its delegators in this model. The validator can make itself be slashed "n" times, with a linear increase in the cost to withdraw for its constituent delegators. It is anticipated that the slashing penalty is sufficiently high that this won't be a practical concern.

2.3 Inflation

Inflation is the idea that we want every staked coin to create more staking tokens as time progresses. The purpose being to drive down the relative worth of unstaked tokens. Each block, every staked token should produce x staking tokens as inflation, where x is calculated from a function $inflation$ which takes state and the block information as input. Let x_i represent the evaluation of $inflation$ in the ith block. The goal of this section is to auto-bond inflation in the fee distribution model without iteration. This is done by preserving the invariant that every state entry contains the rewards one would have if they had bonded one stake at genesis until that corresponding block.

In state a variable should be kept for the number of tokens one would have now due to inflation, given that they bonded one token at genesis. This is $\prod_0^{now}(1+x_i)$. Each period now stores this total inflation product along with what it already stores per-period.

Let R_i be the fee rewards in block i, and n_i be the total amount bonded to that validator in that block. The correct amount of rewards which 1 token at genesis should have now is:

$$Reward(now) = \sum_{i=0}^{now} \left(\prod_{j=0}^{i} 1 + x_j \right) * \frac{R_i}{n_i}$$

The term in the sum is the amount of stake one stake becomes due to inflation, multiplied by the amount of fees per stake.

Now we cast this into the period frame of view. Recall that we build the rewards by creating a state entry for the rewards of the previous period, and keeping track of the rewards within this period. Thus we first define the correct amount of rewards for each successive period, proving correctness of this via induction. We then show that the state entry that gets efficiently built up block by block is equal to this value for the latest period.

Let $start, end$ denote the start/end of a period.

Suppose that $\forall f > 0$, $Reward(end(f))$ is correctly constructed as

$$Reward(end(f)) = Reward(end(f-1)) + \sum_{i=start(f)}^{end(f)} \left(\prod_{j=0}^{i} 1 + x_j \right) \frac{R_i}{n_i}$$

and that for $f = 0$, $Reward(end(0)) = 0$. (With period 1 being defined as the period that has the first bond into it) It must be shown that assuming the supposition $\forall f \leq f_0$,

$$Reward(end(f_0+1)) = Reward(end(f_0)) + \sum_{i=start(f_0+1)}^{end(f_0+1)} \left(\prod_{j=0}^{i} 1 + x_j \right) \frac{R_i}{n_i}$$

Using the definition of $Reward$, it follows that:

$$\sum_{i=0}^{end(f_0+1)} \left(\prod_{j=0}^{i} 1 + x_j \right) * \frac{R_i}{n_i} = \sum_{i=0}^{end(f_0)} \left(\prod_{j=0}^{i} 1 + x_j \right) * \frac{R_i}{n_i} + \sum_{i=start(f_0+1)}^{end(f_0+1)} \left(\prod_{j=0}^{i} 1 + x_j \right) \frac{R_i}{n_i}$$

Since the first summation on the right hand side is $Reward(end(f_0))$, the supposition is proven true. Consequently, the reward for just period f adjusted for the amount of inflation 1 token at genesis would produce, is:

$$\sum_{i=start(f)}^{end(f)} \left(\prod_{j=0}^{i} 1 + x_j \right) \frac{R_i}{n_i}$$

Note that

$$\sum_{i=start(f)}^{end(f)} \left(\prod_{j=0}^{i} 1 + x_j \right) \frac{R_i}{n_i} = \left(\prod_{j=0}^{end(f-1)} 1 + x_j \right) \sum_{i=start(f)}^{end(f)} \left(\prod_{j=start(f)}^{i} 1 + x_j \right) \frac{R_i}{n_i}$$

By definition of period, and inflation being applied every block,
$n_i = n_{start(f)} \left(\prod_{j=start(f)}^{i} 1 + x_j \right)$. This cancels out the product in the summation, therefore

$$\sum_{i=start(f)}^{end(f)} \left(\prod_{j=0}^{i} 1 + x_j \right) \frac{R_i}{n_i} = \left(\prod_{j=0}^{end(f-1)} 1 + x_j \right) \frac{\sum_{i=start(f)}^{end(f)} R_i}{n_{start(f)}}$$

Thus every block, each validator just has to add the total amount of fees (The R_i term) that goes to delegates to some per-period term. When creating a new period, $n_{start(f)}$ can be cached in state, and the product is already stored in the previous periods state entry. You then get the next period's $n_{start(f)}$ from the consensus' power entry for this validator. This is thus extremely efficient per block.

When withdrawing, you take the difference as before, calculating the difference between the reward entry at the withdrawing height and the bonding height. This yields the amount of rewards you would have obtained with $(\prod_0^{begin\ bonding\ period} 1 + x)$ stake from the block you began bonding at until now. $(\prod_0^{begin\ bonding\ period} 1 + x)$ is known, since its included in the state entry for when you bonded. You then divide the entitled fees by $(\prod_0^{begin\ bonding\ period} 1 + x)$ to normalize it to being the amount of rewards you're entitled to from 1 stake at that block to now. Then as before, you multiply by the amount of stake you had initially bonded.

In addition to the above, the withdrawer also needs rewards due to inflation itself. This can be done by taking the accumulated inflation factor, and dividing it by the inflation factor until the beginning of the bonding period. This factor is $(\prod_{begin\ bonding\ period}^{now} 1 + x)$, and then that gets scaled by how much they initially bonded.

The inflation function could vary per block, and per validator if ever a need arose. If the inflation rate is the same for all validators then there can be a single state entry for the entries corresponding to the product of inflations. Inflation creation can trivially be epoched as long as inflation isn't required within the epoch, through changes to the $inflation$ function.

2.4 Withdrawing with no iteration over slashes

Notice that a slash is the same as a negative inflation rate for a validator in one block. For example a 20% slash is equivalent to a -20% inflation for a validator in a block. Given correctness of auto-bonding inflation with different inflation rates per-validator, it follows that handling slashes can be correctly done by simply setting the validators inflation factor in that block to be the negative of the slash factor. This significantly simplifies the withdrawal procedure.

2.5 Auto bonding fees

Auto bonding of fees also follows from the correctness of the inflation model. Split up the rewards into one component with only the staking token, and one component with the remaining tokens. Add to the inflation rate for that block for that validator, $\frac{amount_of_staking_token}{n_i}$, n_i being the validators stake in that block. Set the rewards to then just be the remaining tokens.

2.6 Delegation updates

Updating your delegation amount is equivalent to withdrawing earned rewards and a fully independent new delegation occurring in the same block. The same applies for redelegation. From the view of fee distribution, partial redelegation is the same as a delegation update and a new delegation.

3 State Requirements

State entries can be pruned quite effectively. Suppose for the sake of exposition that there is at most one delegation / withdrawal to a particular validator in any given block. Then each delegation is responsible for one addition to state. Only the next period, and this delegator's withdrawal could depend on this entry. Thus once this delegator withdraws, this state entry can be pruned. For the entry created by the delegator's withdrawal, that is only required by the creation of the next period. Thus once the next period is created, that withdrawal's period can be deleted.

This can be easily adapted to the case where there are multiple delegations / withdrawals per block, by maintaining a reference count in each period starting state entry.

The slash entries for a validator can only be pruned when all of that validator's constituent delegators have their bonding period starting after the slash. This seems ineffective to keep track of, thus it is not worth it. Each slash should instead remain in state until the validator unbonds and all delegators have their fees withdrawn.

Thus, with reference counting, it will always be the case that the total reference count for a particular validator is equal to the number of active delegations (each keeping a reference to the period ended by their delegation) plus the number of slashes (each keeping a reference to the period ended by the slash) plus one (for the most recent period).

4 Implementers Considerations

We have heretofore described F1, a pragmatic fee distribution algorithm with many benefits. The overhead per block is a simple iteration over the bonded validator set, which will often occur anyways due to underlying proof of stake logic (such as to check whether any validators have entered or left). Consequently it can be implemented with minimal additional, as the state entry reads and writes can be cached. All calculations are exact, modulo minor errors resulting from fixed precision decimals. F1 supports arbitrary inflation and fee models, which can vary per validator per block (which enables desirable incentive mechanisms, for example paying only validators which signed the block), as can the commission rates of the individual validators. The simplicity of the scheme lends itself well to implementation. F1 has been implemented in the Cosmos SDK and will be used in the Cosmos Hub blockchain.

References

1 Sunny Aggarwal. Cosmos Proof of Stake. Crypto Economics Security Conference 2018, 2018. URL: https://www.youtube.com/watch?v=XxZ04w2x4nk.

2 Vitalik Buterin. Discouragement Attacks. ETH research, 2018. URL: https://github.com/ethereum/research/blob/367507e0785f488cc269e0a3a61f49ce3c000327/papers/discouragement/discouragement.pdf.

3 Giulia Fanti, Leonid Kogan, Sewoong Oh, Kathleen Ruan, Pramod Viswanath, and Gerui Wang. Compounding of Wealth in Proof-of-Stake Cryptocurrencies. Arxiv, October 2018. arXiv:1809.07468.

Selfish Mining and Dyck Words in Bitcoin and Ethereum Networks

Cyril Grunspan
Léonard de Vinci, Research Center, Paris – La Défense, France
cyril.grunspan@devinci.fr

Ricardo Pérez-Marco
CNRS, IMJ-PRG, Univ. Paris 7, Paris, France
ricardo.perez-marco@imj-prg.fr

Abstract

The main goal of this article is to present a direct approach for the formula giving the long-term apparent hashrates of Selfish Mining strategies using only elementary probabilities and combinatorics, more precisely, Dyck words. We can avoid computing stationary probabilities on Markov chain, nor stopping times for Poisson processes as in previous analysis. We do apply these techniques to other bockwithholding strategies in Bitcoin, and then, we consider also selfish mining in Ethereum.

2012 ACM Subject Classification Mathematics of computing → Markov processes

Keywords and phrases Bitcoin, Blockchain, Ethereum, Proof-of-Work, Selfish Mining, Stubborn Mining, Apparent Hashrate, Revenue Ratio, Catalan Distributions, Dyck Words, Random Walk

Digital Object Identifier 10.4230/OASIcs.Tokenomics.2019.11

1 Introduction

Background

Selfish mining (in short SM) is a non-stop blockwithholding attack described in [1] which exploits a flaw in the Bitcoin protocol in the difficulty adjustment formula [2]. The strategy is made of attack cycles. During each attack cycle, the attacker adds blocks to a secret fork and broadcasts them to peers with an appropriate timing. This is a deviant strategy from the Bitcoin protocol since an honest miner never withholds validated blocks and always mines on top of the last block of the official blockchain [7].

A rigorous profitability analysis that incorporates time considerations was done in [2]. The objective function based on sound economics principles that allows the comparison of profitabilities of different mining strategies with repetition is the *Revenue Ratio* $\frac{\mathbb{E}[R]}{\mathbb{E}[T]}$ where R and T are respectively the revenue and the duration time per attack cycle. A blockcwithholding attack slows down the production of blocks, hurting the profitability per unit time of all miners, including the attacker. Only after a difficulty adjustment, the attack can become profitable. The mean duration time of block production becomes equal to $\mathbb{E}[L] \cdot \tau_B$ where L is the number of blocks added to the official blockchain by the network per attack cycle and $\tau_B = 600$ sec. is the mean validation time of a block in Bitcoin network [4]. For Ethereum τ_E is around 12 sec. (in what follows, we use subscript B or E depending on which network we consider).

The Revenue Ratio becomes proportional to the long-term *apparent hashrate* of the strategy $\tilde{q} = \frac{\mathbb{E}[Z]}{\mathbb{E}[L]}$ where Z is the number of blocks added by the attacker to the official blockchain per attack cycle. This apparent hashrate becomes a proxy for the Revenue Ratio and can be used as a benchmark for profitability, but only after a difficulty adjustment. Several methods have been devised to compute $\tilde{q}$. The original approach from [1] uses a Markov model. Then the stationary probability is computed and used to compute the long term apparent hashrate. In [2] we use Martingale techniques and consider Poisson processes

and associated stopping times in order to compute the Revenue Ratio, and also the expected number of blocks $\mathbb{E}[Z]$ added by the attacker to the blockchain per attack cycle. The Revenue Ratio is computed at once using Doob's Stopping Time Theorem for Martingales. This last method has the advantage to compute the correct profitability analysis directly, not by means of the proxy of the long term apparent hashrate. For example, we can compute how long it takes for the attacker to have profit, something that is impossible to compute with the old Markov chain model. Moreover, with the Martingale techniques we clearly identify the difficulty adjustment formula as the origin of the vulnerability of the protocol. A Bitcoin Improvement Proposal (BIP) was proposed in [2] to prevent blockwithholding attacks. It consists in incorporating orphan blocks in the computation of the apparent hashrate of the network, and this is done by signaling orphan blocks. Something similar is done in Ethereum where rewards are given to some orphan blocks ("uncle" blocks). The goal was to favor mining decentralization.

Main goal

In this article we present a direct combinatorial approach for the direct computation of the apparent hashrate for different blockwithholding strategies in Bitcoin and Ethereum. These formulas are sometimes complicated, so it is remarkable that such a direct approach is possible. We don't need to use Markov chain, nor Martingale theory, and only elementary combinatorics using Dyck words. This analysis does not provide the full strength of the Martingale theory approach, but provides the basic formulas to estimate the long term apparent hashrates, and hence the profitabilities of the different strategies. The situation in Ethereum is combinatorially more complex due to the reward of "uncle" blocks and their signaling, which gives a larger spectrum of possible strategies. Our combinatorial approach also gives closed-form formulas for the apparent hashrate of one of the most effective strategy.

Notation

As usual, the relative hashrate of the honest miners (resp. attacker) is denoted by p (resp. q) and γ is its *connectivity* to the network. We have $p+q=1$, $q<\frac{1}{2}$ and $0\leq\gamma\leq 1$. When a competition occurs between two blocks or two forks, γ is the fraction of the honest miners who mine on top of a block validated by the attacker.

We will make use of Catalan numbers and Dyck words. Catalan numbers can be defined by

$$C_n=\frac{1}{2n+1}\binom{2n}{n}=\frac{(2n)!}{n!(n+1)!}$$

and their generating series is

$$C(x)=\sum_{n=0}^{+\infty}C_n x^n=\frac{1-\sqrt{1-4x}}{2x}$$

A Dyck word is a string (word) composed by two letters X and Y such that no initial segment of the string contains more Y's than X's. The relation with Catalan numbers is that the n-th Catalan number is the number of Dyck words of length $2n$. We refer to [6] for more properties and background material.

2 Selfish mining

An attack cycle for the SM strategy (see [2]) can be described as a sequence $X_0 \dots X_n$ with $X_i \in \{S, H\}$. The index i indicates the i-th block validated since the beginning of the cycle and the letter S, resp. H, indicates that the selfish, resp. honest, miner has discovered this block. From this labelling we will get the relation with Dyck words.

▶ **Example 1.** The sequence SSSHSHH means that the selfish miner has been first to validate three blocks in a row, then the honest miners have mined one, then the selfish miner has validated a new one and finally the honest miners have mined two blocks. At this point, the advantage of the selfish miner is only of one block. So according to the SM strategy, he decides to publish his whole fork and ends his attack cycle. In that case, we have $L = Z = 4$.

We are interested in the distribution of the random variable L.

▶ **Theorem 2.** *We have* $\mathbb{P}[L = 1] = p, \mathbb{P}[L = 2] = pq + pq^2$ *and for* $n \geqslant 3$, $\mathbb{P}[L = n] = pq^2(pq)^{n-2}C_{n-2}$ *where* C_n *is the* n*-th Catalan number.*

Proof. For $n \geqslant 3$, we note that $\{L = n\}$ is a collection of sequences of the form $w = \mathrm{SSX}_1 \cdots X_{2(n-2)}H$ with $X_i \in \{S, H\}$ for all i, such that if S and H are respectively replaced by the brackets "(" and ")" then, $X_1 \cdots X_{2(n-2)}$ is a Dyck word (i.e. balanced parenthesis) with length $2(n-2)$ (see [6]). The number of letters "S" (resp. "H") in w is n (resp. $n-1$). So, we get $\mathbb{P}[L = n] = p^{n-1}q^nC_{n-2}$ (see [6]). Finally, from the observation that $\{L = 1\} = \{H\}, \{L = 2\} = \{\mathrm{SSH}, \mathrm{SHS}, \mathrm{SHH}\}$, the result follows. ◀

▶ **Corollary 3.** *We have* $\mathbb{E}[L] = 1 + \frac{p^2q}{p-q}$

Proof. This formula results from the well know relations from [3]

$$\Sigma_{n\geqslant 0} p(pq)^n C_n = 1 \tag{1}$$

$$\Sigma_{n\geqslant 0} np(pq)^n C_n = \frac{q}{p-q} \tag{2}$$

◀

We can now compute the apparent hashrate.

▶ **Theorem 4.** *The long-term apparent hashrate of the selfish miner in Bitcoin is*

$$\tilde{q}_B = \frac{[(p-q)(1+pq)+pq]q - (p-q)p^2q(1-\gamma)}{pq^2+p-q}$$

Proof. When $L \geqslant 3$ we are in the situation where all blocks validated by the selfish miner end-up in the official blockchain. So, we have $Z = L$. If $L = 1$, then we have $Z = 0$. Moreover, we have $Z(\mathrm{SSH}) = Z(\mathrm{SHS}) = 2$ and $Z(\mathrm{SHH}) = 0$ (resp. 1) with probability $1-\gamma$ (resp. γ). So, we have

$$\begin{aligned}\mathbb{E}[Z] &= \mathbb{E}[L] - p - p^2q\gamma - 2p^2q(1-\gamma)\\ &= \mathbb{E}[L] - (p + p^2q + p^2q(1-\gamma))\end{aligned}$$

Using Corollary 3 we get,

$$\begin{aligned}\frac{\mathbb{E}[Z]}{\mathbb{E}[L]} &= \frac{p^2q + p - q - (p-q)(p + p^2q + p^2q(1-\gamma))}{pq^2+p-q}\\ &= \frac{[(p-q)(1+pq)+pq]q - (p-q)p^2q(1-\gamma)}{pq^2+p-q}\end{aligned}$$

This is Proposition 4.9 from [2] which is another form of Formula (8) from [1]. ◀

3 Stubborn Mining

We consider now two other block withholding strategies described in [8].

3.1 Equal Fork Stubborn Mining

In this strategy, the attacker never tries to override the official blockchain but, when possible, he broadcasts the part of his secret fork sharing the same height as the official blockchain as soon as the honest miners publish a new block. The attack cycle ends when the attacker has been caught-up and overtaken by the honest miners by one block [3, 8]. We show that the distribution of $L-1$ is what we defined as a (p,q)-Catalan distribution of first type in [3].

▶ **Theorem 5.** *For $n \geqslant 0$ we have $\mathbb{P}[L = n+1] = p(pq)^n C_n$.*

Proof. For $n \geqslant 0$, $\{L = n+1\}$ corresponds to sequences of the form $w = X_1 \cdots X_{2n} H$ with $X_i \in \{S, H\}$ for all i, such that if S and H are respectively replaced by the brackets "(" and ")" then, $X_1 \cdots X_{2n}$ is a Dyck word with length $2n$. ◀

▶ **Corollary 6.** *We have $\mathbb{E}[L] = \frac{p}{p-q}$*

Proof. Follows from (1) and (2). ◀

▶ **Theorem 7.** *The long-term apparent hashrate of a miner following the Equal-Fork Stubborn Mining strategy is given by*

$$\tilde{q} = \frac{q}{p} - \frac{(1-\gamma)(p-q)}{\gamma p}(1 - pC((1-\gamma)pq))$$

Proof. In an attack cycle, all the honest blocks except the last one have a probability γ to be replaced by the attacker. So, we have $\mathbb{E}[Z|L = n+1] = n + 1 - \frac{1-(1-\gamma)^{n+1}}{\gamma}$ (see Lemma B.1 in [3]). Conditioning by $\{L = n+1\}$ for $n \in \mathbb{N}$ and using Theorem 5, we get

$$\mathbb{E}[Z] = \frac{q}{p-q} - \frac{1-\gamma}{\gamma}(1 - pC((1-\gamma)pq))$$

and the result follows. ◀

3.2 Lead Stubborn Mining

This strategy is similar to the selfish mining strategy but this time the attacker takes the risk of being caught-up by the honest miners. When this happens, there is a final competition between two forks sharing the same height. when the competition is resolved, a new attack cycles starts. In this case, the distribution of $L-1$ turns out to be a (p,q)-Catalan distribution of second type as defined in [3].

▶ **Theorem 8.** *We have $\mathbb{P}[L = 1] = p$ and for $n \geqslant 1$, $\mathbb{P}[L = n+1] = (pq)^n C_{n-1}$.*

Proof. We have $\{L = 1\} = \{H\}$ and for $n \geqslant 0$, the condition $\{L = n+1\}$ corresponds to sequences of the form $w = \mathrm{S}X_1 \cdots X_{2(n-1)}\mathrm{HY}$ with $X_1, \ldots, X_{2(n-1)}, Y \in \{S, H\}$ and such that if S and H are respectively replaced by the brackets "(" and ")" then, $X_1 \cdots X_{2(n-1)}$ is a Dyck word with length $2(n-1)$. ◀

▶ **Corollary 9.** *We have $\mathbb{E}[L] = \frac{p-q+pq}{p-q}$*

Proof. Follows from (1) and (2). ◀

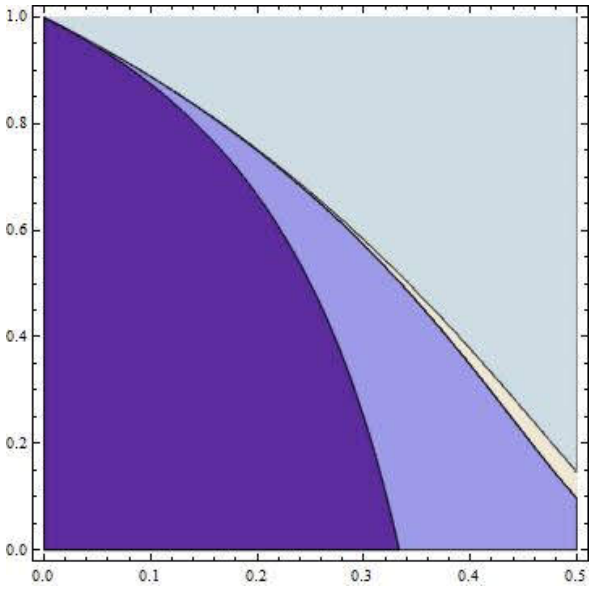


Figure 1 From left to right: HM, SM, LSM, EFSM.

By repeating the same argument as in the proof of Theorem 7 for the computation of $\mathbb{E}[Z]$, we obtain the following theorem [3].

▶ **Theorem 10.** *The long-term apparent hashrate of a miner following the Lead Stubborn Mining strategy is given by*

$$\tilde{q} = \frac{q(p+pq-q^2)}{p+pq-q} - \frac{pq(p-q)(1-\gamma)}{\gamma} \cdot \frac{1-p(1-\gamma)C((1-\gamma)pq)}{p+pq-q}$$

We plot regions in the parameter space $(q,\gamma) \in [0,0.5] \times [0,1]$ according to which strategy is more profitable. We get Figure 1 [3] (HM honest mining, SM selfish mining, LSM Lead Stubborn mining, EFSM Equal Fork Stubborn mining).

4 Selfish mining in Ethereum

Ethereum is a cryptocurrency based on a variation of the GHOST protocol [11]. The reward system is different than in Bitcoin, and this introduces a supplementary complexity in the analysis of block withholding strategies. Contrary to Bitcoin, mined orphan blocks can be rewarded like regular blocks, with a reward smaller than regular blocks. The condition for an orphan block to get a reward is to be an "uncle" referred by a "nephew" which is "not too far". By definition, an "uncle" is a stale block whose parent belongs to the main chain and a "nephew" is a regular block which refers to this "uncle". "Not too far" means that the distance d between the uncle and the nephew is less than some parameter value n_1. The distance is the number of blocks which separates the nephew to the uncle's parent in the main chain. When this situation occurs, the nephew gets an additional reward of πb and the uncle gets a reward $K_u(d)b$ where b denotes the coinbase in Ethereum. Today's parameter values are $n_1 = 6$, $K_u(d) = \frac{8-d}{8} \cdot \mathbf{1}_{1 \leqslant d \leqslant 6}$, $\pi = \frac{1}{32}$ and $b = 2$ ETH [9].

There is little published research on selfish mining in Ethereum except for [9] and [10] based on numerical simulations. In [9], through a Markov chain approach, a non-closed infinite double sum is given for the apparent hash rate of the attacker.

The general study of selfish mining in Ethereum is complex because equivalent selfish mining strategies in Bitcoin are no longer equivalent for Ethereum. The attacker can choose to refer or not uncle blocks. Referring uncle blocks provides an extra revenue but hurts the main goal of selfish mining of lowering the difficulty. Also, he can choose to create artificially more uncles by broadcasting the part of his secret fork sharing the same height as the public blockchain of the honest miners. All this different strategies are analyzed in [5]. In the present article we restrict to a couple of strategies.

In the strategy studied in [9], the attacker creates as many uncles as possible and tries to refer all of them. In [5], we prove that this strategy is not optimal and is less profitable than the strategy we study in this article, for which we obtain a closed form formula for the apparent hashrate of the attacker using only elementary combinatorics.

In the strategy we consider, the attacker never broadcasts his fork, which remains secret until he is on the edge of being caught-up by the honest miners or is actually caught up (this last case can only occur when the attack cycle starts with SH). In addition, the attacker always refers to all possible uncle blocks.

We denote by R the revenue by cycle of the selfish miner following this strategy. We have $R = R_s + R_u + R_n$ where R_s is the revenue coming from "static" blocks in the main chain i.e., $R_s = Zb$, R_u is the revenue coming from uncles and R_n is the additional revenue coming from nephews.

▶ Remark 11. We always have $R_u = 0$ except when the attack cycle is SHH and the last block mined by the honest miners has been mined on top of an honest block. In that case, the first block mined by the selfish miner is referred by the second block of the honest miners.

It follows from this remark that

$$\mathbb{E}\left[\frac{R_u}{b}\right] = p^2q(1-\gamma)K_u(1) \tag{3}$$

It remains to compute $\mathbb{E}[R_n]$.

▶ **Definition 12.** *If ω is an attack cycle, we denote by $U(\omega)$ (resp. $U_s(\omega)$, $U_h(\omega)$) the random variable counting the number of uncles created during the cycle ω which are referred by nephew blocks (resp. nephew blocks mined by the selfish miner, nephew blocks mined by the honest miners) in the cycle ω or in a later attack cycle.*

We denote by $V(\omega)$ the random variable counting the number of uncles created during the cycle ω and are referred by nephew blocks (honest or not) in an attack cycle strictly after ω.

▶ **Proposition 13.** *We have $\mathbb{E}[U] = q - q^{n_1+1}$.*

Proof. We have $U = 0$ if and only if the attack cycle is H or if it starts with $n_1 + 1$ blocks of type S. Otherwise, we have $U = 1$. So, $\mathbb{E}[U] = \mathbb{P}[U > 0] = 1 - (p + q^{n_1+1}) = q - q^{n_1+1}$ ◀

We compute now $\mathbb{E}[V]$

▶ **Proposition 14.** *We have $\mathbb{E}[V] = pq^2 \cdot \frac{1-(pq)^{n_1-1}}{1-pq}$.*

Proof. We have $V = 1$ if and only if the attack cycle ω is SS..SH..H with $2 \leq k \leq n_1$ S. In that case, the first block H is an uncle that will be referred by the first future official block in the attack cycle after ω. Otherwise, $V = 0$. So, $\mathbb{E}[V] = pq^2 + \ldots + p^{n_1-1}q^{n_1}$, and we get the result. ◀

▶ **Proposition 15.** *We have $\mathbb{E}[U_h] = p^2q + \big(p + (1-\gamma)p^2q\big)\, pq^2 \cdot \frac{1-(pq)^{n_1-1}}{1-pq}$.*

Proof. Let ω be an attack cycle and let ω' be the attack cycle after ω. If $U_h^{(1)}(\omega)$ (resp. $U_h^{(2)}(\omega)$) counts the number of uncles referred by honest nephews only present in ω (resp. in ω'), then we have $U_h = U_h^{(1)} + U_h^{(2)}$. Moreover, $U_h^{(1)}(\omega) = \mathbf{1}_{\omega=\text{SHH}}$ and $U_h^{(2)}(\omega) = \mathbf{1}_{\omega'\in E}\cdot V(\omega)$ where E is the event that ω' is either H or SHH with a second honest block mined on top of the first honest block. Hence we get the result by taking expectations since ω and ω' are independent. ◀

▶ **Corollary 16.** *We have*

$$\mathbb{E}\left[\frac{R_n}{\pi}\right] = q^2(1+p) - q^{n_1+1} - \big(p+(1-\gamma)p^2q\big)\,pq^2\cdot\frac{1-(pq)^{n_1-1}}{1-pq} \tag{4}$$

Proof. We have $\mathbb{E}[U_s] = \mathbb{E}[U] - \mathbb{E}[U_h]$ and we use Proposition 13 and Proposition 15. ◀

We can now compute the apparent hashrate of the selfish miner in Ethereum. We have two cases to consider: The old difficulty adjustment formula (similar to the one in Bitcoin), and the current difficulty adjustment formula that takesinto account referred uncles.

▶ **Theorem 17.** *The long term apparent hashrate $\tilde{q}_{E,0}$ of the selfish miner in Ethereum with its old difficulty adjustment formula is given by $\tilde{q}_{E,0} = \tilde{q}_B + \tilde{q}_u K_u(1) + \tilde{q}_n\pi$ with*

$$\tilde{q}_u = \frac{p^2q(1-\gamma)(p-q)}{p-q+p^2q}$$
$$\tilde{q}_n = \frac{(p-q)\left(q^2(1+p) - q^{n_1+1} - \big(p+(1-\gamma)p^2q\big)\,pq^2\cdot\frac{1-(pq)^{n_1-1}}{1-pq}\right)}{p-q+p^2q}$$

The long term apparent hashrate $\tilde{q}_E$ of the selfish miner in Ethereum with its current difficulty adjustment formula is

$$\tilde{q}_E = \tilde{q}_{E,0}\cdot\xi$$

where

$$\xi = \frac{p-q+p^2q}{p^2q+(p-q)\big(1+q-q^{n_1+1}\big)}$$

Proof. We have $\tilde{q}_{E,0} = \frac{\mathbb{E}[R]}{\mathbb{E}[L]}$ and $\tilde{q}_E = \frac{\mathbb{E}[R]}{\mathbb{E}[L]+\mathbb{E}[U]}$ We then use Proposition 13, (3), (4) and the formula for $\tilde{q}_B$ in Theorem 4. ◀

We can now compare this strategy to selfish mining in Bitcoin. Observe that $\tilde{q}_{E,0} > \tilde{q}_B$, where $\tilde{q}_B$ is the long term apparent hashrate of the Bitcoin selfish miner. Therefore, the minimal threshold $q_{\min}$ such that the inequality $\tilde{q} > q$ for $q > q_{\min}$ is always lower in Ethereum with its old adjustment formula than in Bitcoin. This is due to the particular reward system that indeed favors selfish mining as we have proved. Notice also that when $q > q_{\min}$, the attack is profitable faster in Ethereum than in Bitcoin because of another difference in the protocols: In Ethereum the difficulty is updated at each block and in Bitcoin only after 2016 blocks.

Figure 2 plots the regions in parameter space $(q,\gamma) \in [0,0.5]\times[0,1]$ where each strategy HM or SM is more profitable. We find $q_{\min} \approx 9.5\%$ when $\gamma = 0$.

Now, Ethereum with its new difficulty adjustment formula is more resilient to selfish mining. Figure 3 plots the region in parameter space $(q,\gamma) \in [0,0.5]\times[0,1]$ where each strategy HM or SM is more profitable.

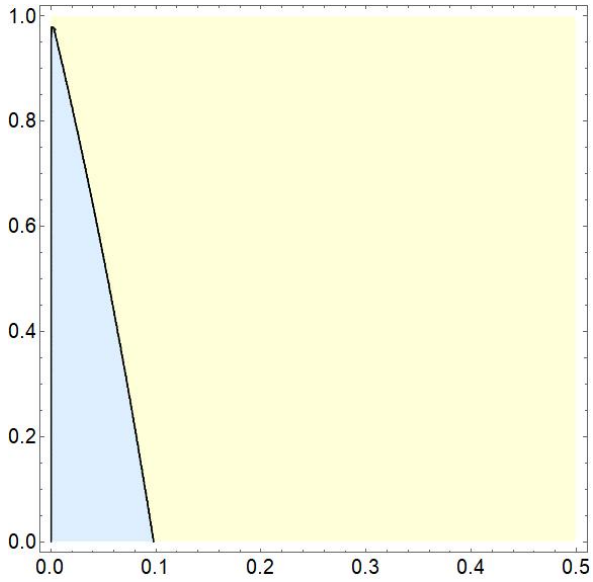


Figure 2 HM vs. SM Ethereum old difficulty adjustment.

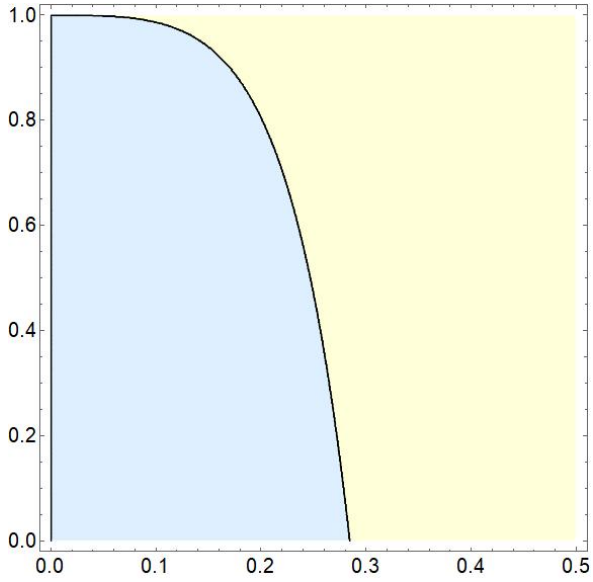


Figure 3 HM vs. SM Ethereum new difficulty adjustment.

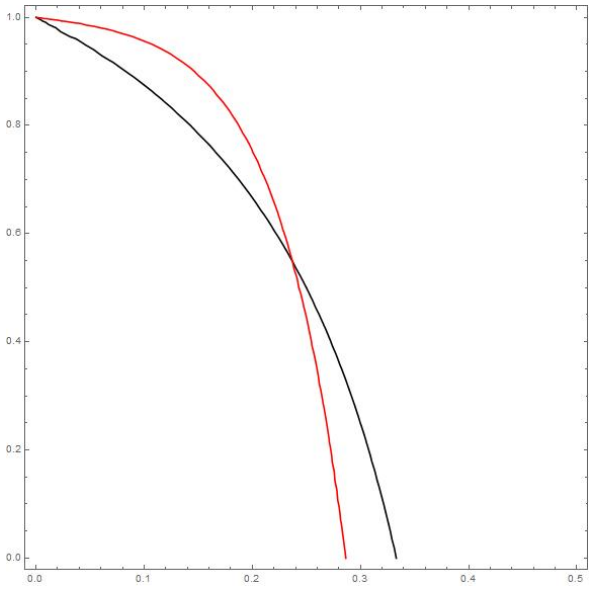


Figure 4 HM vs. SM in Bitcoin and Ethereum.

We note that Bitcoin is more resilient to selfish mining when the relative hashrate of the attacker is high, but we have the opposite for smaller relative hashrates. This means that when the relative hashrate of the attacker is small (resp. high) then, the connectivity of the attacker should be higher (resp. lower) in Ethereum than in Bitcoin for the attack to be profitable. Figure 4 compares the thresholds curves between HM and SM in Bitcoin and Ethereum.

5 Conclusions

We have computed closed-form formulas for the long term apparent hashrate of different blockwithholding strategies for Bitcoin and Ethereum using only elementary combinatorics, Dyck words, Catalan numbers, and their properties. Although this approach does not provide a complete analysis of the profitability of the strategies, as for example the time it takes to the strategy to become profitable, this minimalist approach is sufficient to compare profitabilities in the long run. In the strategies studied we have show the impact of the different reward system. For these strategies, depending on given parameters (q, γ), relative hashrate and connectivity of the attacker, we have determined which network is more resilient to selfish mining attacks.

References

1 I. Eyal and E. G. Sirer. Majority is not enough: bitcoin mining is vulnerable. *Commun. ACM*, 61(7):95–102, 2018. doi:10.1145/3212998.

2 C. Grunspan and R. Pérez-Marco. On profitability of Selfish mining. *arXiv*, 2018. arXiv:1805.08281.

3 C. Grunspan and R. Pérez-Marco. On profitability of Stubborn mining. *arXiv*, 2018. arXiv:1808.01041.

4 C. Grunspan and R. Pérez-Marco. On profitability of Trailing mining. *arXiv*, 2018. arXiv:1811.09322.

5 C. Grunspan and R. Pérez-Marco. Selfish mining in Ethereum. *arXiv*, 2019. arXiv:1904.13330.

6 T. Koshy. *Catalan numbers with applications*. Oxford University Press, Oxford, 2009.

7 S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System, 2008. URL: `https://bitcoin.org/bitcoin.pdf`.

8 K. Nayak, S. Kumar, A. Miller, and E. Shi. Stubborn Mining: Generalizing Selfish Mining and Combining with an Eclipse Attack. In *IEEE European Symposium on Security and Privacy, EuroS&P 2016, Saarbrücken, Germany, March 21-24, 2016*, pages 305–320. IEEE, 2016. `doi:10.1109/EuroSP.2016.32`.

9 J. Niu and C. Feng. Selfish Mining in Ethereum. *arXiv*, 2019. `arXiv:1901.04620`.

10 F. Ritz and A. Zugenmaier. The Impact of Uncle Rewards on Selfish Mining in Ethereum. In *2018 IEEE European Symposium on Security and Privacy Workshops, EuroS&P Workshops 2018, London, United Kingdom, April 23-27, 2018*, pages 50–57. IEEE, 2018. `doi:10.1109/EuroSPW.2018.00013`.

11 Y. Sompolinsky and A. Zohar. Secure High-Rate Transaction Processing in Bitcoin. In Rainer Böhme and Tatsuaki Okamoto, editors, *Financial Cryptography and Data Security - 19th International Conference, FC 2015, San Juan, Puerto Rico, January 26-30, 2015, Revised Selected Papers*, volume 8975 of *Lecture Notes in Computer Science*, pages 507–527. Springer, 2015. `doi:10.1007/978-3-662-47854-7_32`.

B-CoC: A Blockchain-Based Chain of Custody for Evidences Management in Digital Forensics

Silvia Bonomi
Research Center of Cyber Intelligence and Information Security (CIS),
Department of Computer, Control, and Management Engineering "A. Ruberti",
Sapienza Università di Roma, Via Ariosto 25, 00145 Rome, Italy
https://www.cis.uniroma1.it
bonomi@diag.uniroma1.it

Marco Casini
Department of Computer, Control, and Management Engineering "A. Ruberti",
Sapienza Università di Roma, Via Ariosto 25, 00145 Rome, Italy
casini.1724011@studenti.uniroma1.it

Claudio Ciccotelli
Research Center of Cyber Intelligence and Information Security (CIS),
Department of Computer, Control, and Management Engineering "A. Ruberti",
Sapienza Università di Roma, Via Ariosto 25, 00145 Rome, Italy
https://www.cis.uniroma1.it
ciccotelli@diag.uniroma1.it

Abstract

One of the main issues in digital forensics is the management of evidences. From the time of evidence collection until the time of their exploitation in a legal court, evidences may be accessed by multiple parties involved in the investigation that take temporary their ownership. This process, called *Chain of Custody* (CoC), must ensure that evidences are not altered during the investigation, despite multiple entities owned them, in order to be admissible in a legal court. Currently digital evidences CoC is managed entirely manually with entities involved in the chain required to fill in documents accompanying the evidence. In this paper, we propose a Blockchain-based Chain of Custody (B-CoC) to dematerialize the CoC process guaranteeing auditable integrity of the collected evidences and traceability of owners. We developed a prototype of B-CoC based on Ethereum and we evaluated its performance.

2012 ACM Subject Classification Applied computing → Computer forensics; Applied computing → Evidence collection, storage and analysis

Keywords and phrases Digital Forensics, Chain of Custody, Digital Evidence, Private Blockchain, Ethereum

Digital Object Identifier 10.4230/OASIcs.Tokenomics.2019.12

Related Version A technical report is available at https://arxiv.org/abs/1807.10359

Acknowledgements This work has been partially supported by the Sapienza Ateneo 2017 project INOCS.

1 Introduction

One of the main issues in digital forensics is the management of evidences. From the time of evidence collection until the time of their exploitation in a legal court, evidences may be accessed by multiple parties involved in the investigation that take temporarily their ownership. The *Chain of Custody* is the process of validating how any kind of evidence has been gathered, tracked and protected on its way to a court of law. Chain of Custody (CoC) is not a mandatory step in forensic analysis. However, it is extensively used as evidences,

to be acceptable in a court or in legal procedures, must be proved to be not altered during investigations. Thus, a good CoC process should use a standard for dealing and handling evidences (digital or not), regardless of whether the evidence will be used in a trial or not.

The main requirements of a CoC process are:

- **Integrity**: the evidence has not been altered or corrupted during the transferring.
- **Traceability**: the evidence must be traced from the time of its collection until it is destroyed.
- **Authentication**: all the entities interacting with an evidence must provide an irrefutable sign as a recognizable proof of their identity.
- **Verifiability**: the whole process must be verifiable from every entity involved in the process.
- **Security – Tampering proof**: Changeovers of an evidence cannot be altered or corrupted.

Currently, CoC process requirements are met by employing a physical handover of evidences where, at each step, documents are filled in and signed in front of officers. In this paper, we take a step toward the dematerialisation of this process by proposing a Blockchain-based architecture for CoC of digital evidences called *B-CoC*. Leveraging on the features offered by blockchain technologies, we defined an architecture able to support the CoC process. To this aim, we proposed an architecture, namely B-CoC, that is able to realise an Evidence log with integrity checks (i.e., every process is able to verify and detect if there has been an integrity breach that would invalidate the digital evidence). B-CoC integrates together an ordinary database with a permissioned blockchain: the first represents the *Evidence DB* where digital evidences are stored, while the second represents the *Evidence Log* that allows to track digital evidences during their lifecycle. This distinction is done to store each type of information in the most suited kind of distributed storage: digital evidences are quite static and large piece of information and do not need particular support for updates while the evidence log is characterised by a reduced size of record to be stored and is subjected to a high update frequency.

In particular, we set up a private permissioned blockchain and we implemented a smart contract to keep track of the ownership changes during the evidence lifecycle. We implemented our prototype on an Ethereum [9] private network and we evaluated the impact of the system configuration parameters on performance.

2 Background

2.1 Blockchain technology

The blockchain technology implements a decentralized fully replicated append-only ledger in a peer-to-peer network, originally employed for the Bitcoin cryptocurrency [7]. All participating nodes maintain a full local copy of the blockchain. The blockchain consists of a sequence of blocks containing the transactions of the ledger. Transactions inside blocks are sorted chronologically and each block contains a cryptographic hash of the previous block in the chain. Nodes create new blocks as they receives transactions, which are broadcast in the network. Once a block is complete, they start the consensus process to convince other nodes to include it in the blockchain. In the original blockchain technology employed in Bitcoin the consensus process is based on *Proof-of-Work* (PoW) [7]. With PoW nodes compete with each other in confirming transactions and creating new blocks by solving a mathematical puzzle. While solving a block is a computational intensive task, verifying its validity is easy. To incentivize such mechanism, solving a block also results in mining a certain amount of

bitcoins, which is the reward for block creators (usually referred to as *miners*). Sometimes, more than one miner may generate a valid block thus creating forks in the chain. Forks are solved by accepting only the longest branch as the valid continuation of the chain (thus eliminating forks eventually). The main advantage of PoW, over traditional consensus algorithms, is that an attacker would have to control the majority of the computational power of the network, rather than the majority of the nodes, which is considered more difficult and virtually impossible in public large-scale networks.

The main criticism to PoW is its huge demand of energy, which also prevents its applicability in certain contexts. This has led to the investigation of alternative forms of consensus for the blockchain, such as *Proof-of-Stake* [5]. With PoS, a set of nodes, called *validators*, take turns proposing new blocks and voting on them. Validators put a stake in the network (e.g., a given amount of cryptocurrency) and are incentivized to act honestly so as not to lose the stake. Indeed, the blockchain keeps track of the set of validators, which are ousted if they behave maliciously (thus losing their stake).

A specific type of PoS is *Proof-of-Authority* (PoA) in which individual's identity (rather than cryptocurrency) is at stake. With PoA validators must have been preventively authorized and their identities are known. Thus, acting maliciously results in losing personal reputation and ultimately in being expelled from the validator set.

While PoW is particularly suited for *public* networks, both PoS and PoA may be suitable for *private* networks (where PoW would probably fail short as it would be much easier to control the majority of the computational power). Moreover, PoW and PoS can be used in *permissionless* networks, that is, networks where nodes can freely join the network without previous authorization (e.g., as in Bitcoin and Ethereum). PoA, on the other hand, is typically employed in *permissioned* blockchain networks, that is, networks in which nodes cannot freely join and become validators, but rather they have to be preventively authorized.

2.2 Ethereum and Smart Contracts

Etherium [9] can be seen as a decentralized virtual machine based on the blockchain technology. The Ethereum Virtual Machine (EVM) runs programs, referred to as *smart contracts*, whose state is stored in the Ethereum blockchain. Every node execute a local EVM. When an account wants to execute a function of a smart contract, it issues a transaction which is broadcast to the network. Each node executes the transaction on its local EVM and stores it, along with the new computed state, in the blockchain.

In Ethereum each EVM instruction consumes a virtual resource referred to as *gas*.

Gas can be seen as the fuel of the EVM and is employed to incentivize miners to execute transactions and include them in the blockchain. Indeed, for each transaction, miners are rewarded by the issuer with the payment of fees proportional to the total amount of gas "consumed" to execute that transaction.

To prevent mined blocks from becoming too large, which may severely impact block propagation and processing latency, each block has a *block gas limit*, which is the maximum amount of gas all transactions included in the block are allowed to consume. Thus, an issued transaction may not be included in the current block by a miner because it would exceed the block gas limit. In such case, the issued transaction would have to wait until the next block creation.

The public Ethereum blockchain (often referred to simply as "Ethereum") is a public permissionless networks which adopts PoW as consensus algorithm (even though it is planned to switch to PoS in the future). However, all major Ethereum implementations [1, 3] allow to configure many aspects of the protocol, such as the actual consensus algorithms employed, and allow to build custom public/private permissionless/permissioned blockchain networks.

2.3 Istanbul BFT consensus protocol

Istanbul Byzantine Fault Tolerance (IBFT) [2] is an adaptation of the Practical Byzantine Fault Tolerance (PBFT) [6] algorithm to serve as a PoA consensus algorithm for the Ethereum protocol. IBFT can tolerate at most f faulty validators out of a total of $n = 3f + 1$ validators. The IBFT algorithm proceeds in rounds with a new block created every T seconds, where the *block period* T is a constant configuration parameter. In each round one of the validators is elected as the *proposer*. The proposer creates the new block and broadcasts it to all validators with a *pre-prepare* message. Upon receiving pre-prepare messages, validators enter the *pre-prepared* phase and broadcast *prepare* messages. This, ensures that validators are aligned to the same round and block. Upon receiving $2f + 1$ prepare messages, validators enter the *prepared* phase and broadcast *commit* messages to inform other validators that they accept the proposed block. Finally, upon receiving $2f + 1$ commit messages, validators enter the *committed* phase and insert the block in the blockchain.

3 System Model

CoC model. A digital evidence (or electronic evidence) is any probative information stored or transmitted in digital form that a party may use in a trial to a court case. Digital evidences are collected by authorised parties (usually police officers) that become their temporary (first) owners.

For the sake of presentation and without loss of generality, in the following we will consider a single digital evidence d_ev collected by an authorised entity e_0 that holds its ownership. During investigations, several authorised entities (e.g., police offices, lawyers, judges, magistrates, etc.) may need to access, acquire and/or own temporarily d_ev. The set of authorised entities that can interact with d_ev is denoted with A_{d_ev}. Each authorised entity has a unique identifier known to all and he/she owns credentials that allows him/her to be authenticated and take actions in the CoC process.

At each time t, d_ev can have just one *owner* and the owner must belong to A_{d_ev}. If an authorised entity e_i needs to acquire and own d_ev, the current owner needs to issue a *transfer* request towards e_i. The change of ownership happens if and only if $e_i \in A_{d\_ev}$ and the transfer record is written permanently in the *evidence log*.

Network Model. The system is composed by a set of processes $p_1, p_2, \ldots, p_n$, one for each authorised entity in A_{d_ev}. Each process p_i has a pair of private-public keys that it uses to authenticate itself and to sign messages. Processes are connected trough a peer-to-peer network (authenticated perfect links). We consider that authorised entities are trusted but up to f of them (with $n > 3f$) can be compromised and controlled by an adversary i.e., they may behave as a Byzantine process deviating arbitrarily from the protocol.

4 B-CoC Architecture

The Blockchain-based Chain of Custody (B-CoC) architecture proposed in this paper is based on a *private* and *permissioned* blockchain. This choice has been driven by the *authentication* requirement of the CoC process that does not allow unauthorised and untrusted parties to manage digital evidences and thus to be in the network.

As shown in Figure 1, B-CoC is composed mainly of three components: (i) the *Evidences DB*, (ii) the *Evidence Log* and the (iii) *Frontend* interface. The Evidence DB is an ordinary database and/or file repository where we store the actual digital evidences, while CoC related

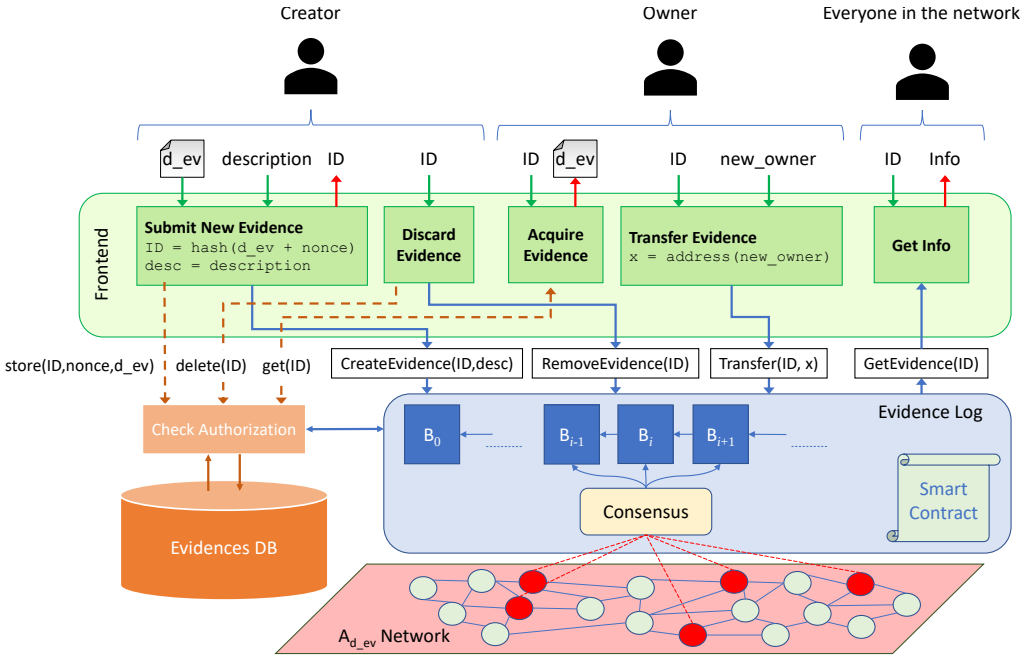


Figure 1 B-CoC architecture.

data are stored in the Evidence Log, which is implemented through the blockchain technology. The reason for this separation is twofold. First of all, evidences can be too large to be efficiently stored in the blockchain (for example, an evidence may be a bit-by-bit copy of a storage device of several TBs of capacity). Secondly, and most importantly, if evidences were stored in the blockchain, every node in the blockchain network would have access to them, while only authorized nodes should be allowed to acquire an evidence. Therefore, we store in the blockchain only the information regarding the CoC process and an hash of the evidence which allows to verify evidences integrity during acquisition.

Evidence DB. The Evidences DB is an ordinary distributed database and/or file repository where the original digital evidence is stored along with an identifier `ID`, obtained as the hash of the evidence and a nonce (to guarantee uniqueness of `ID`s). This database is distributed and is managed by trusted entities (e.g., law court officers). Moreover, each access is executed only if the requesting entity is authorized to perform such access according to its role.

Evidence Log. The Evidence Log is implemented trough the blockchain technology and stores, for each evidence, its `ID`, a description, the identity of the submitter (which we call *creator*) and the complete history of owners up to the current one, including the time at which changes of ownership occurred. Note that while the evidence itself is not stored in the blockchain, the `ID` allows to verify that the evidence has not been tampered with, provided that a robust cryptographic hash function is used to generate it.

The evidence log is implemented on top of a peer-to-peer network composed by all authorised entities. Such network can be decomposed in two sets of nodes:

- *Validator nodes*: they have mainly the following functionalities: (i) storing a copy of the blockchain, (ii) validating transactions and (iii) create, propose and add blocks to the chain (i.e., participate to the consensus protocol). This is the set of nodes that must be preventively authorized with the role of validators in the permissioned blockchain.

- *Lightweight nodes*: they can be seen as clients of the chain since they simply issue transactions and need to rely on validators for adding and validating their transactions.

Taking Italy as a use case, each validator may correspond to the main coordinator of the court of one of the 20 regional capitals. Lightweight nodes, instead, would represent all the other involved parties such as police departments, forensic investigators, forensic consultants and so on. The Evidence Log runs a smart contract which exposes four primitives (see Figure 1):

- `CreateEvidence(ID, description)`: stores a new evidence entry in the blockchain with the specified `ID` and `description`, setting the submitter identity as the *creator* and current *owner* of the evidence.
- `Transfer(ID, newowner)`: transfers the ownership of an evidence (registering the handover). It fails if the issuer is not the current owner.
- `RemoveEvidence(ID)`: removes an evidence entry. It fails if the issuer is not the creator.
- `GetEvidence(ID)`: returns the information in the evidence entry. Namely, the `ID`, description, creator and all owners with the time of each change of ownership.

Implementation details of the Evidence Log and the smart contract are discussed in Section 5.

Frontend Interface. The frontend represents the interface between B-CoC and its users. A local instance runs on each node and interacts with the Evidences DB and the Evidence Log (through a local blockchain node). When an authorized user submits a new digital evidence *d_ev* to the system, he/she takes the role of *creator* of *d_ev* (see Figure 1). The frontend generates the `ID` for *d_ev* using a nonce n, sends the command `store(ID, n, d_ev)` to the Evidence DB and issues the `CreateEvidence()` transaction in the Evidence Log. As already discussed the submitter is also registered as the first owner in the blockchain. When the *Check Authorization* component of the Evidence DB receives the `store(ID, n, d_ev)` command, it starts to monitor the Evidence Log for the corresponding `CreateEvidence()` transaction. Only upon confirmation that this transaction has been inserted in the Evidence Log, the Check Authorization component actually stores the pair (ID, n, d_ev) into the Evidence DB.

The creator of an evidence *d_ev* can request to discard it from the system (e.g., because it is no more legally valid). If he/she is authorized to do so, the corresponding entry is removed from the Evidence Log by issuing the `RemoveEvidence()` transaction. If the transaction succeeds, the corresponding evidence is deleted from the Evidence DB by issuing the `delete` command. Upon receiving the `delete` command the Check Authorization component of the Evidence DB checks if the corresponding `RemoveEvidence()` transaction has been inserted in the Evidence Log. If the transaction is not present, the `delete` command fails and sends an error response to the frontend.

When a user wants to acquire an evidence *d_ev*, the Frontend sends a request to the Evidences DB which will serve the request only if the user is the current owner of *d_ev*. This check is performed by the Check Authorization component by interacting with the Evidence Log.

The change of ownership of an evidence *d_ev* is performed by issuing a `Transfer()` transaction specifying the new owner. Note that this operation does not involve the Evidence Log in any way.

Finally, every user in the B-CoC network can query the Evidence Log to get the entry of an evidence (which contains all relevant information except the evidence itself). This is performed by simply issuing the `GetEvidence()` transaction.

5 Evidence Log Implementation

As described in Section 4, B-CoC Evidence Log is designed as a private and permissioned blockchain. The blockchain infrastructure is implemented through *Geth* [1] a popular implementation of a full Ethereum node. Geth allows to setup a private network and configure all aspects of the blockchain and the consensus protocol employed. Given the design of a private permissioned blockchain we adopt a PoA-based consensus. Namely, the IBFT consensus protocol described in Section 2.3. Let us note that, at the time of our development, IBFT was the only Byzantine tolerant consensus protocol available using Geth. On top of this blockchain infrastructure, we run a smart contract implementing the CoC process. The choice of implementing B-CoC using Geth has been driven by a cost-benefit analysis. We considered several blockchain technologies, namely full Ethereum, Geth with PoA consensus and Hyperledger Fabric, and we evaluated them against the requirement of our application. None of them is currently matching perfectly our needs but we believe that Geth with PoA consensus is the most appropriate given its ease of adaptation, deployment and complexity. The implementation of B-CoC Evidence Log involves three steps: (i) the initialization of the private blockchain, (ii) the creation of the private network and (iii) the creation and deployment of the smart contract.

5.1 Private chain initialization

The setup of a new blockchain involves the creation of its *genesis* block. This is the first block of a blockchain and contains the initial parameters. The only configuration parameters that are of interest for the purposes of the following discussion are:

- *Block Period T*: the block period of the IBFT consensus algorithm (see section 2.3);
- *Block Gas Limit G*: Maximum amount of gas transactions in a block are allowed to consume (see section 2.2);
- *Validators*: The Ethereum addresses of the pre-authorized validators.

The genesis block is used to initialize each node of the network.

5.2 Private network setup

First of all, to build the private peer-to-peer network we need to setup the peer discovery service to allow new nodes to enter the network and know other nodes. This is accomplished with the `bootnode` tool (of the Geth tools suite). This tool allows to run special nodes (with known IP addresses) that validators and lightweight nodes will contact when first started to exchange peer information.

Validators and lightweight nodes are Geth nodes. First, we configure the set of validators (which is fixed and known in advance) with the genesis block and we run them through the `geth` command (of the Geth tools suite). Validators are created once at the beginning and they never leave the network, unless they act maliciously and are expelled. Lightweight nodes, instead, can be created and join/leave the network at any time. They are created with the `geth` tool as well, but their addresses are not included in the genesis block.

5.3 Smart contract implementation

The smart contract has been implemented through the Solidity contract-oriented programming language [4]. Due to space constraints, the code of the smart contract is reported in the Appendix. The smart contract manages entries associated to digital evidences (i.e., the entries of the Evidence Log). Each `Evidence` entry (lines 3-10) consists of the `ID`, the Ethereum

Table 1 Size and gas used by each transaction.

TX	size(TX) (bytes)	gas(TX) (units)
`CreateEvidence(0)`	207	170207
...	...	...
`CreateEvidence(1024)`	1233	897367
`Transfer()`	174	80502
`RemoveEvidence()`	142	236478

address of the creator, the address of the owner, a string field to store the description of the evidence and two arrays `taddr` and `ttime` that store, respectively, the evidence handovers and the times at which they occurred. These arrays are chronologically sorted from the creator to the current owner. All evidence items are stored in a map indexed by evidence `IDs` (line 11). The smart contract has a total of four functions implementing the primitives of the Evidence Log described in Section 4. The `CreateEvidence(ID, description)` function creates a new `Evidence` entry with the specified `ID` and `description`, and the address of the related transaction sender as the creator and current owner of the evidence (line 26). The `Transfer(ID, newowner)` function transfers the ownership of the evidence identified by `ID` to the entity identified by the address `newowner` (line 35). Note that only the current owner of an evidence can transfer ownership (`OnlyOwner` modifier). The `RemoveEvidence(ID)` function removes an evidence from the map of evidences (line 41). No further operations can be performed on a removed evidence. Note that only the creator of an evidence can remove the evidence (`OnlyCreator` modifier). The `GetEvidence(ID)` function returns all fields of an evidence entry (line 46).

Note that while calling the first three functions results in issuing transactions to the blockchain that modify the state of the smart contract, the `GetEvidence` function only returns an entry and does not modify the state. In the context of the Solidity language these are called *constant* functions or *views*. Calling views does not result in the issuing of transactions, but rather they are executed locally by the node's local EVM.

6 Evaluation

In this section we evaluate how the parameters of B-CoC, namely the block period T and the block gas limit G, affect its performance. This analysis allows to guide the choice of the most appropriate configuration parameters in each scenario, as discussed in Section 7. Section 6.1 reports an analysis of the transaction latency, Section 6.2 evaluates the space overhead due to block headers and Section 6.3 discusses the growth rate of the blockchain.

Notation. In the following sections we will use the notation TX to refer to a *transaction type* (i.e., a non-constant function of a smart contract) and tx to refer to an *execution of a transaction*. For example, TX may refer to the transaction type `Transfer()`, while tx may refer to an actual execution of a `Transfer()` of an evidence. We will use gas(tx) and size(tx) to indicate, respectively, the gas consumed by the execution of a transaction tx and the size (in bytes) of tx when included in a block. Note that, in general, both gas(tx) and size(tx) depend on the particular execution of tx. In practice, for our smart contract, transaction types `Transfer()` and `RemoveEvidence()` have constant size and gas used, while for `CreateEvidence()` such parameters depend exclusively on the length ℓ of

the `description` parameter. Thus, for ease of presentation we will consider a different transaction type `CreateEvidence(`ℓ`)` for each value of ℓ. Since we limit the length of the `description` parameter to 1024 characters, we consider 1025 different transaction types ($\ell = 0, \ldots, 1025$). Thus, each transaction type has constant size and constant consumed gas and, therefore, we will consider $\text{size}(TX) = \text{size}(tx)$ where tx is an execution of TX and use the two members of the equation interchangeably, as well as $\text{gas}(TX) = \text{gas}(tx)$. We will refer to $\mathcal{SC} = \{TX_1, \ldots, TX_n\}$ as the set of transaction types of the smart contract. Table 1 reports the size and gas of the transaction types in our smart contract. Due to space constraints Table 1 only shows `CreateEvidence(`ℓ`)` for $\ell = 0$ and $\ell = 1024$, but the size and gas used by such transaction types increase with ℓ.

6.1 Transaction latency

The transaction latency $L(tx) = L_{\text{B}}(tx) + L_{\text{C}}(b)$ is the time elapsed from the issue of the transaction to its inclusion in the blockchain. It is the sum of the *block inclusion latency* $L_{\text{B}}(tx)$, that is the time required by tx to be included in a block b of the current proposer, and the *consensus latency* $L_{\text{C}}(b)$, which is the time required to reach consensus on block b and include it in the blockchain: In the next two sections we will analyze these two terms separately.

6.1.1 Block inclusion latency

The block inclusion latency $L_{\text{B}}(tx)$ is the time required for a transaction tx to be included in a block. Indeed, whenever a new transaction is issued it may not *fit* in the block of the current proposer due to the block gas limit G. In such case, the transaction is reissued in the next block period.

More formally, let $\text{block}(tx)$ be the block in which, eventually, transaction tx is included, and $\text{time}(tx)$, $\text{time}(b)$ be, respectively, the time at which tx is issued and the time at which a block b was created (i.e., the beginning of b's block period), then:

$$L_{\text{B}}(tx) = \text{time}(\text{block}(tx)) + T - \text{time}(tx) \tag{1}$$

where T is the block period.

The block inclusion latency is affected by the block period parameter T, the gas limit G and the workload, i.e., the rate of transactions issued to the system in the unit of time. Suppose that we are able to precisely characterize the workload the system is subject to and to set G such that every issued transaction is included in the block of the current proposer. In such ideal conditions, $L_{\text{B}}(tx) \in [0, T]$. That is, the maximum block inclusion latency is the block period T. Clearly, setting $G = \infty$ would meet the ideal conditions for every possible workload volume, but on the other side would negatively impact consensus latency, as blocks could increase indefinitely (see next section). Thus, we would like to set G as small as possible (to reduce consensus latency), but large enough so that (at least on average) every transaction is included in the block of the current proposer. Thus the ideal value of G depends on the workload. However, rather than the number of transactions per seconds, it depends on the *gas rate*, i.e., the amount of gas consumed by the transactions issued in a given block period. Indeed, the minimum value of G that guarantees the ideal conditions for the block inclusion latency is the maximum gas rate.

Figure 2 shows the results of three experiments that confirm the previous claim. In each experiment we set a different value of the block gas limit (G_1, G_2 G_3) and we progressively increased the gas rate from the start to the end of the experiment. From the figure we

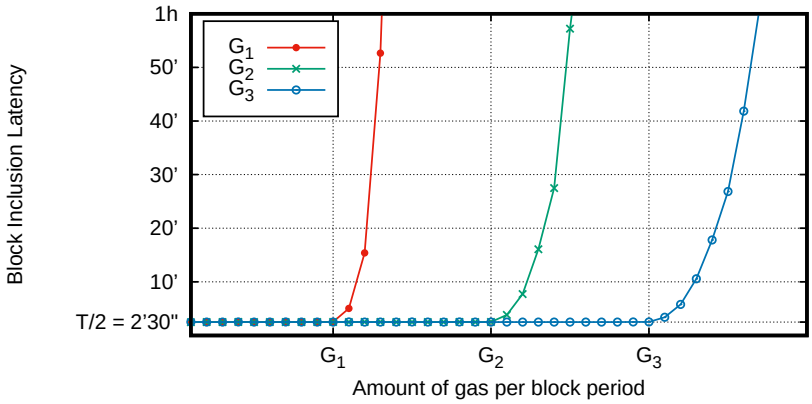


Figure 2 Mean block inclusion latency varying the *gas rate*, i.e., the amount of gas consumed by transactions in a block period.

can clearly see that, in each experiment, when the gas rate is less than or equal to the gas limit the average block inclusion latency is approximately equal to the expected value of $T/2$ (because transactions are issued uniformly distributed in each block period) while the maximum latency is T (not shown in the figure). However, as soon as the gas rate exceeds the block gas limit the average block inclusion latency starts increasing indefinitely as expected.

This analysis provides a lower bound for the value of the block gas limit (i.e., the maximum gas rate), that allows to minimize the maximum block inclusion latency to T. However, determining such value may be difficult. Section 7 reports a more general and detailed discussion on setting the parameters of B-CoC.

6.1.2 Consensus latency

Given the consensus protocol described in section 2.3, the consensus latency, i.e., the time required to propagate a block b between the validators and reach consensus, can be approximated by the formula $L_{\mathrm{C}}(b) \approx \frac{s_{\mathrm{PP}}(b)+s_{\mathrm{P}}+s_{\mathrm{C}}}{R}$, where $s_{\mathrm{PP}}(b)$ is the size of the *pre-prepare* message, s_{P} is the size of the *prepare* message, s_{C} is the size of the *commit* message and R is the bandwidth of the slowest communication channel between two validators nodes (bytes/sec). While s_{P} and s_{C} are constant, the pre-prepare message piggybacks the block b and thus $s_{\mathrm{PP}}(b)$ depends on size(b). Since R is typically a constant that depends on the infrastructure connecting the validator nodes, the only factor that we can adjust to control the latency is the size of a block.

The size of a block is the sum of the size of the transactions in it plus the size of the block header s_{H} (which is constant). In our prototype implementation of B-CoC, $s_{\mathrm{H}} = 1909$ bytes. The actual number and type of transactions in a block depends on many factors, including the block period T, the block gas limit G and ultimately the particular set of transactions sent during a given time period. Thus, in general, different blocks have different sizes. However, we can control the maximum block size $S^{\max}$, and thus the maximum consensus latency $L_{\mathrm{C}}^{\max}$, by adjusting the block gas limit G.

For a given value of G, the maximum block size $S^{\max}$ can be computed by solving the following optimization problem:

▶ **Problem 1** (UKP).

$$\begin{aligned} \text{maximize} \quad & \sum_{TX_i \in \mathcal{SC}} \text{size}(TX_i) \cdot x_i \\ \textit{subject to} \quad & \sum_{TX_i \in \mathcal{SC}} \text{gas}(TX_i) \cdot x_i \leq G \\ & x_i \in \mathbb{N}, \quad i = 1, \ldots, n \end{aligned}$$

where x_i is the number of times a transaction of type TX_i appears in the block of maximum size. The optimal solution $\{x_1^*, \ldots, x_n^*\}$ leads to the maximum block size:

$$S^{\max} = s_{\text{H}} + \text{OPT}_{\text{UKP}}(G) = s_{\text{H}} + \sum_{TX_i \in \mathcal{SC}} \text{size}(TX_i) \cdot x_i^*$$

Problem 1 is an instance of the well-known unbounded knapsack problem [8], where transaction types correspond to the items to fit in the knapsack, while transactions' size and consumed gas correspond, respectively, to items' value and weight. The block gas limit parameter G corresponds to the knapsack maximum weight.

While the general unbounded knapsack problem is NP-hard (with time complexity $O(nG)$), this particular instance turns out to be trivial. Indeed, it is easy to see that `Transfer()` *dominates* all other transaction types [8]. That is, given any block containing at least a transaction tx of type in $\mathcal{SC} \setminus \{\texttt{Transfer()}\}$, we can always replace tx with a sufficient number of `Transfer()` so as to obtain a better solution to Problem 1. For example, we can always replace a transaction of type `RemoveEvidence()` with a single `Transfer()` and obtain a solution that consumes less gas but have larger size. The same occurs if we replace `CreateEvidence(0)` with 2 `Transfer()`, or `CreateEvidence(1024)` with at least 8 `Transfer()`. This implies that the optimal solution of this instance of the unbounded knapsack problem corresponds to a block consisting of only `Transfer()` transactions. Therefore, let $g_{\text{T}} = \text{gas}(\texttt{Transfer()})$, $s_{\text{T}} = \text{size}(\texttt{Transfer()})$, the solution is simply given by:

$$S^{\max} = s_{\text{H}} + \left\lfloor \frac{G}{g_{\text{T}}} \right\rfloor \cdot s_{\text{T}} \tag{2}$$

Note that $S^{\max}$ cannot be an arbitrary integer, but only one such that $S^{\max} = s_{\text{H}} + k \cdot s_{\text{T}}$, $k \in \mathbb{N}$.

Once chosen the appropriate value of $S^{\max}$ (one that allows to limit the maximum consensus latency to an acceptable bound), G can be set to any value such that:

$$G = \frac{S^{\max} - s_{\text{H}}}{s_{\text{T}}} \cdot g_{\text{T}} + r = k \cdot g_{\text{T}} + r, \qquad r = 0, \ldots, g_{\text{T}} - 1 \tag{3}$$

Equations 2 and 3 allows to determine an upper-bound for G so as to bound the maximum consensus latency $L_{\text{C}}^{\max}$. A more general discussion on how to properly set B-CoC's parameters is reported in section 7.

6.2 Block headers overhead

As discussed in section 6.1.1, the block period T affects transactions block inclusion latency. A longer block period implies higher latency. On the other hand, a shorter period results in a higher number of blocks created per time interval (since a block for each block period is

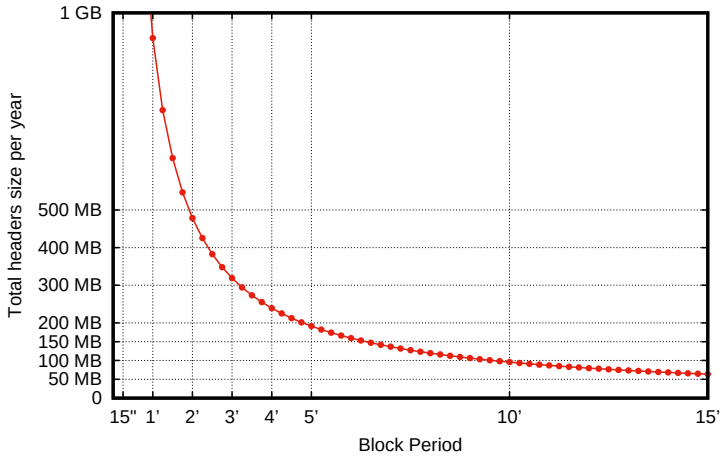


Figure 3 Total headers size per year for different block periods T.

created). Since each block has a fixed size header, the larger the number of blocks created, the higher the space occupied by block headers compared to transactions in the blockchain, that is, the higher the space overhead.

The headers size overhead, i.e., the total size of block headers, at any time t is:

$$\mathrm{OH}(t) = s_{\mathrm{H}} \cdot \frac{t}{T} \tag{4}$$

Note that this value only depends on the number of blocks in the chain at time t, and not on the number of transactions. Figure 3 shows the space overhead per year ($s_{\mathrm{H}} = 1909$ bytes in our prototype implementation), that is, how much blockchain's space is taken up by block headers every year. For example, for $T = 5$ minutes the space overhead is around 191 MB per year. We find this value of T a good trade-off between transaction latency and block headers overhead for this particular application of the blockchain.

6.3 Blockchain growth rate

The blockchain can be seen as an append-only database. That is, its size cannot shrink over time. If $I_{\mathcal{SC}}(t)$ is the set of transactions included in the blockchain at time t, then the blockchain total size at time t is:

$$\mathrm{size}_{\mathrm{bc}}(t) = s_{\mathrm{g}} + \mathrm{overhead}_{\mathrm{bc}}(t) + \sum_{tx \in I_{\mathcal{SC}}(t)} \mathrm{size}(tx)$$

where s_{g} is the size of the genesis block. Therefore, the growth rate over a time interval $[t_1, t_2]$ is $\mathrm{size}_{\mathrm{bc}}(t_2) - \mathrm{size}_{\mathrm{bc}}(t_1)$, that is:

$$\mathrm{GR}(t_1; t_2) = s_{\mathrm{H}} \cdot \frac{t_2 - t_1}{T} + \sum_{tx \in I_{\mathcal{SC}}(t_1; t_2)} \mathrm{size}(tx) \tag{5}$$

where $I_{\mathcal{SC}}(t_1; t_2) = I_{\mathcal{SC}}(t_2) \setminus I_{\mathcal{SC}}(t_1)$.

Obviously, how fast a blockchain grows over time depends mainly on the transaction rate. Another factor that affects the growth rate is the block period T. As already shown in section 6.2, this parameter affects the headers overhead and thus the first term of equation 5. The block gas limit parameter G may also affect the growth rate, as, if not properly

Table 2 Growth rate for different classes of workloads (n `CreateEvidence(1024)`, n `RemoveEvidence()`, $10n$ `Transfer()` per year).

Workload	GR − OH	GR with $T = 5'$	OH / GR (%)
$n = 10000$	29.7 MB/year	221.08 MB/year	86.56%
$n = 100000$	297 MB/year	488.45 MB/year	39.18%
$n = 1000000$	2.9 GB/year	3.09 GB/year	6.05%

dimensioned, it may increase latency spreading the incoming transaction rate over a larger time period, thus, decreasing the growth rate (i.e., G would affect the number and type of transactions included in $I_{\mathcal{SC}}(t_1; t_2)$ and thus the second term of equation 5). However, the analysis detailed in section 6.1, should allow to set the value of G so as to bound transaction latency. In practice, G should be set greater than the average gas rate to avoid an ever increasing latency. In this conditions, if the growth rate is computed over a large enough interval of time (to hide the effects of potential peak gas rate periods), the block gas limit parameter should not affect the growth rate significantly (that is, if properly set, G should not affect $I_{\mathcal{SC}}(t_1; t_2)$). Otherwise, G should be set to a larger value.

By using equation 5 we computed the annual growth rate for different classes of workloads. Since we were not able to find any publicly available statistics about evidence collection and transfer, we considered different classes of synthetic workloads with n new evidence creations and removals and $10n$ transfers per year. The results of this analysis are reported in Table 2. The second column of Table 2 reports the annual growth rate without considering the headers size overhead, while the third one includes the overhead term computed for $T = 5'$. Finally, the fourth column shows the overhead percentages. Even in presence of a very large number of evidence collection (1 million per year) and transfers (10 millions per year) the growth rate is around 3 GB per year, which seems acceptable given the capacities of todays storage devices.

7 Discussion on the configuration of the parameters

Section 6 discusses how the parameters of B-CoC affect its performance with respect to different aspects, namely the transaction latency, the block headers overhead and the blockchain growth rate. Here we give a comprehensive discussion on how to set B-CoC parameters appropriately.

7.1 Setting the block period T

The block period T affects transactions block inclusion latency (see section 6.1.1) and the block headers overhead (section 6.2), that ultimately affects the blockchain growth rate. As already discussed in section 6.2, a shorter block period results in a lower maximum block inclusion latency, but also in a higher block header overhead. To find the best trade-off one can use equation 4. For example, with our prototype implementation of B-CoC we consider $T = 5'$ to be a good trade-off between latency and block header overhead. Indeed, any further increase of T would result in a small improvement in terms of overhead reduction compared to the increase of latency (as shown in Figure 3).

7.2 Setting the block gas limit G

The block gas limit affects both term of the transaction latency. In particular in section 6.1, we describe an analysis that allows to derive a lower bound G_{L} for G, to limit block inclusion latency and an upper bound G_{U} to limit consensus latency.

When $G_L \leq G_U$ it is safe to set G equal to any value in $[G_L, G_U]$ to obtain a maximum block inclusion latency bound by T and the desired maximum consensus latency. On the other hand, if $G_L > G_U$, it is not possible to have both terms of transaction latencies bounded by the desired values. In such case, one should set G to the best trade-off between block inclusion latency and consensus latency. A good strategy may be to discard the lower bound G_L in favor of a new lower bound G_L^{avg} which is set to the average gas rate rather than the maximum gas rate. Setting $G = G_L^{avg}$ would result in block inclusion latency bounded by T on average, with possible periods of increasing latencies, e.g., during peak loads. In this case, if $G_L^{avg} \leq G_U$ one should set $G = G_U$, otherwise $G = G_L^{avg}$. Indeed, setting a value of G less than the average gas rate would result in ever increasing transaction latencies.

8 Conclusion

This paper presented B-CoC, a blockchain-based architecture to dematerialise the CoC process in digital forensics. We also provided a prototype of the B-CoC architecture based on the Geth implementation of Ethereum nodes. Based on the performance evaluation, B-CoC showed to be an effective support for the CoC process as it is able to sustain realistic workload with an acceptable overhead in terms of memory used to store the chain.

The current implementation assumes that the set of validators node is fixed and that validators are available to sacrifice their privacy when participating in the consensus process. As a future work, we are investigating how it is possible to manage a dynamic set of validators and most important we are studying alternatives that allow to increase the level of privacy for validators not altering other dependability and security attributes.

References

1 Geth. `https://github.com/ethereum/go-ethereum/wiki/geth`. [Online; accessed 30-May-2018].

2 Istanbul BFT. `https://github.com/ethereum/EIPs/issues/650`. [Online; accessed 17-July-2018].

3 Parity. `https://parity.io`. [Online; accessed 20-July-2018].

4 Solidity. `https://solidity.readthedocs.io`. [Online; accessed 11-June-2018].

5 Iddo Bentov, Ariel Gabizon, and Alex Mizrahi. Cryptocurrencies Without Proof of Work. In *Financial Cryptography Workshops*, volume 9604 of *Lecture Notes in Computer Science*, pages 142–157. Springer, 2016.

6 Miguel Castro and Barbara Liskov. Practical Byzantine Fault Tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, OSDI '99, pages 173–186, Berkeley, CA, USA, 1999. USENIX Association. URL: `http://dl.acm.org/citation.cfm?id=296806.296824`.

7 Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. URL: `http://www.bitcoin.org/bitcoin.pdf`.

8 Vincent Poirriez, Nicola Yanev, and Rumen Andonov. A hybrid algorithm for the unbounded knapsack problem. *Discrete Optimization*, 6(1):110–124, 2009. `doi:10.1016/j.disopt.2008.09.004`.

9 Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger, 2014.

A Smart Contract Code

```
pragma solidity ^0.4.22;
contract ChainOfCustody {
  struct Evidence {
    bytes32 ID;
    address owner;
    address creator;
    string description;
    address[] taddr;
    uint[] ttime;
  }
  mapping(bytes32 => Evidence) private evidences;

  modifier OnlyOwner(bytes32 ID) {
    require(msg.sender == evidences[ID].owner); _;}
  modifier OnlyCreator(bytes32 ID) {
    require(msg.sender == evidences[ID].creator); _;}
  modifier EvidenceExists(bytes32 ID, bool mustExist) {
    bool exists = evidences[ID].ID != 0x0;
    if (mustExist)
      require(ID != 0x0 && exists);
    else
      require(!exists);
    _;}

  function CreateEvidence (bytes32 ID, string description)
      public EvidenceExists(ID, false) {
    evidences[ID].ID = ID;
    evidences[ID].owner = msg.sender;
    evidences[ID].creator = msg.sender;
    evidences[ID].description = description;
    evidences[ID].taddr.push(msg.sender);
    evidences[ID].ttime.push(now);
  }
  function Transfer(bytes32 ID, address newowner)
      public OnlyOwner(ID) EvidenceExists(ID, true) {
    evidences[ID].owner = newowner;
    evidences[ID].taddr.push(newowner);
    evidences[ID].ttime.push(now);
  }
  function RemoveEvidence(bytes32 ID)
      public OnlyCreator(ID) EvidenceExists(ID, true) {
    delete evidences[ID];
  }
  function GetEvidence(bytes32 ID)
      view public returns(bytes32, address, address,
        string, address [], uint []) {
    return(evidences[ID].ID, evidences[ID].owner,
      evidences[ID].creator, evidences[ID].description,
      evidences[ID].taddr, evidences[ID].ttime);
  }
}
```

Listing 1 Smart contract code.

MixEth: Efficient, Trustless Coin Mixing Service for Ethereum

István András Seres[1]
Eötvös Loránd University, Hungary
http://istvanseres.web.elte.hu/
istvanseres@caesar.elte.hu

Dániel A. Nagy
Eötvös Loránd University, Hungary
daniel@ethereum.org

Chris Buckland
King's College London, United Kingdom
cpbuckland88@gmail.com

Péter Burcsi
Eötvös Loránd University, Hungary
bupe@inf.elte.hu

Abstract

Coin mixing is a prevalent privacy-enhancing technology for cryptocurrency users. In this paper, we present MixEth, which is a trustless coin mixing service for Turing-complete blockchains. MixEth does not rely on a trusted setup and is more efficient than any proposed trustless coin tumbler. It requires only 3 on-chain transactions at most per user and 1 off-chain message. It achieves strong notions of anonymity and is able to resist denial-of-service attacks. Furthermore the underlying protocol can also be used to efficiently shuffle ballots, ciphertexts in a trustless and decentralized manner.

2012 ACM Subject Classification Theory of computation → Cryptographic protocols

Keywords and phrases Cryptography, Verifiable shuffle, Anonymity, Cryptocurrency, Ethereum, Coin mixer, State Channel

Digital Object Identifier 10.4230/OASIcs.Tokenomics.2019.13

Supplement Material https://github.com/seresistvanandras/MixEth

Funding The research at Eötvös Loránd University was partially supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.2-16-2017-00012).
Chris Buckland: is supported by an Ethereum Foundation scaling grant and an Ethereum Community Fund grant.

Acknowledgements We would like to thank Liam Horne for helping with the state channel implementation, Barry Whitehat, Dmitry Khovratovich and Sina Mahmoodi for the insightful comments and discussions.

1 Introduction

Bitcoin [20] and other cryptocurrencies are pseudonymous. Users' public keys are used as pseudonyms in these systems. Transactions essentially record a flow of cryptocurrency from one (or more) public keys to another public key (or more). Flow of cryptocurrency can be easily tracked due to the open and transparent nature of cryptocurrencies' transaction ledger.

[1] Corresponding author: István András Seres, istvanseres@caesar.elte.hu

Moreover, coherent public keys, which are used by the same user, can be clustered merely by analyzing the ledger. Recently several tools and algorithms were proposed to diminish users' privacy [17, 19, 18]. Such deanomyzation attacks are extremely harmful to user privacy, especially in the case when any of the users' pseudonyms, public keys, are linked to their real world identity.

One of the methods to increase users' privacy is coin mixing or tumbling. This technique provides *k-anonymity* or *plausible deniability*. The idea is that k users deposit 1 coin each and then in the course of a coin shuffling protocol either a centralized trusted third party or a smart contract mixes the coins and redistributes them to designated fresh public keys. This powerful technique gives users superior privacy and anonimity since their new received coins cannot be linked to them.

Several coin mixing protocols were proposed in the literature both centralized [6, 26, 13] and decentralized [14, 25, 1, 16, 4]. A major drawback of centralized coin mixing is that the availability of the tumbler is entirely dependent on the trusted party and in most cases theft prevention cannot be guaranteed [6, 26]. On the other hand decentralized tumblers achieve availability, theft prevention and satisfy strong notions of anonymity although they are considerably heavier computationally. In the following we will solely focus on the problem of coin mixing on Ethereum [27].

There is no doubt that there exists a tremendous need for privacy overlays for Ethereum as new tools for transaction deanonymization are getting developed and used [8]. This need of the Ethereum community for privacy was spectacularly embodied in September, 2017 when for several days 68% of all the transaction volume was controlled by a centralized coin mixing service [22].

The two major techniques to provide decentralized mixing services for Ethereum are Möbius, a ring-signature-based solution [16] and Miximus, a zkSNARK-based proposal [1]. Both of them burn tremendous amounts of gas to withdraw funds, which could be prohibitive for many use cases. Möbius requires $335{,}714n$ gas (n is the ring size) while Miximus consumes 1,903,305 gas to verify a zkSNARK proof [2]. As the Ethereum network is congested, ie. blocks were full during 2018[2], we argue that it is essential for the network scalability to aim to create protocols and applications that burn as few gas as possible.

Even though (Ethereum) users and transactions can be deanonymized already on the network layer [21], we consider network anonymity an orthogonal problem to that of anonymity on the transaction ledger.

Our contributions. In this paper, we present a trustless and efficient mixing protocol for Turing-complete blockchains. This protocol can be used to shuffle ciphertexts, ballots or public keys. The protocol have many use cases: shuffling ElGamal-ciphertexts, decentralized mixnets, e-voting.

To show the practicality of the protocol, we introduce MixEth, a privacy-enhancing protocol and a practical tool for Ethereum, to overcome the above mentioned efficiency issues of Ethereum-based coin-mixers while retaining strong notions of anonymity, mixer availability and theft prevention already achieved by previous proposals [16, 1]. MixEth requires as few off-chain messages and on-chain transactions as Möbius and Miximus, meanwhile it burns significantly less gas.

The intuition behind MixEth is to apply Neff's verifiable shuffles [23] in the context of coin mixing. Participants of the tumbler shuffle their public keys in order to break links

[2] `https://etherscan.io/chart/gasused`

between sender and recipient public keys. The key insight is that verifying on-chain a Neff proof about the correctness of a shuffle would be too gas-inefficient, therefore we require receivers to be online to issue fraud proofs, if and only if an incorrect shuffle was made. Whenever recipients consider that enough shuffling was executed, they can withdraw their funds from the mixer.

We also implement MixEth in a state channel to leverage the scalability and instant finality of off-chain scaling solutions. Furthermore, the MixEth protocol could be used in any state channel application to mix funds before going back on-chain.

2 Background

In this section we introduce the building bocks required to create our mixing protocol and MixEth, the trustless coin tumbler.

2.1 Notations

In most cases if it is possible we will stick to the notations used in [16] for sake of uniformity. Let [] denote the empty tuple. For a tuple $t = (x_1, \ldots, x_n)$ we denote as $t[x_i]$ the value stored at x_i. The cardinality of a finite set X is denoted as $|X|$. In the following let $\lambda \in \mathbb{N}$ be the security parameter and its unary representation is 1^λ. If x is uniformly randomly sampled from a set A we write $x \xleftarrow{\$} A$. The symmetric group of degree n is written as S_n. In a cyclic group $\mathbb{G}$, the standardized generator is denoted as G and we use the additive notation. Secret keys and public keys are denoted as sk and pk respectively (or often times s and sG), while the user the corresponding key belongs to is indicated in subscript. Let PK_i denote the set of public keys belonging to receivers at a particular shuffling round i.

We use games in definitions and proofs of security. At the end of each game, the main procedure of game G outputs a single bit. $\Pr(\mathsf{G})$ denotes the probability that the output is 1.

2.2 Cryptographic keys in Ethereum

Ethereum uses Elliptic Curve Cryptography (ECC) to secure users' funds. More specifically, it uses the secp256k1 curve, the same one as used in Bitcoin. If a user wants to create an Ethereum address, first they needs to generate a secret key $s \xleftarrow{\$} \mathbb{Z}_n$, where n is the order of secp256k1 over a finite prime field $\mathbb{F}_p$. The corresponding public key will be sG. Note that any multiples of G is also a generator of curve points since n, the order of the group is also a prime. Accounts in Ethereum are identified by their addresses which can be obtained by taking the right most 20 bytes of the Keccak hashed public key [27].

2.3 Verifiable shuffle

Neff introduced the notion of verifiable shuffle [23]. It is a cryptographic protocol allowing a party to verifiably shuffle a sequence of k modular integers. The output of the shuffle is another k modular integers multiplied by the same secret multiplier only known to the shuffler. The shuffler can generate a publicly verifiable zero-knowledge proof to convince the public that the shuffle was done correctly without disclosing the secret multiplier.

Neff's mathematical construct is extremely powerful, since it only relies on the intractability of the Decision Diffie-Hellman (DDH) problem. Therefore, Neff's verifiable shuffle can also be applied in groups over elliptic curves.

Verifiable shuffle can be used to shuffle a set of public keys, $PK = (s_1G, s_2G \ldots, s_kG)$. Note that secret keys are not known to the shuffler.

1. Shuffler commits to $C = cG$, publishes

$$PK^* = (c(s_{\pi^{-1}(1)}G), c(s_{\pi^{-1}(2)}G), \ldots, c(s_{\pi^{-1}(k)}G))$$

   where π is a random permutation. Shuffler additionally computes and publishes a zero-knowledge proof about the correctness of the shuffle. This proof can be made non-interactive via the Fiat-Shamir heuristic. Let us call C as the shuffling constant.
2. Assuming the proof verifies users gain new public keys with respect to another generator element, namely cG.

For verifying the proof one needs to compute $8k + 5$ exponentiations, however later this result was ameliorated to $3,5k$ exponentiations by Bayer and Groth [3].

So far verifiable shuffles were only applied in voting schemes, we argue that they are useful in trustless coin mixers as well. The key insight in order to be able to apply verifiable shuffles in a decentralized, computational-resource-constrained environment, for instance Ethereum smart contracts, is to dismiss the proof generation for the correctness of the shuffle, rather we request users to give more succinct proofs for the incorrectness of the shuffle, if applicable.

2.4 Decision Diffie-Hellman Problem and Chaum-Pedersen Protocol

The Decision Diffie-Hellman assumption (DDH) is a standard cryptographic hardness assumption which underlies the security of many cryptographic protocols. Roughly speaking DDH states that no efficient algorithm can distinguish between the two distributions (aG, bG, abG) and (aG, bG, cG), where $a, b, c \xleftarrow{\$} \mathbb{Z}_{|\mathbb{G}|}$. It is believed that the DDH assumption holds for elliptic curves with prime order over a prime field with large embedding factor [5], specifically DDH holds for the secp256k1 curve, which is used to generate accounts and sign transactions in Bitcoin and Ethereum among other cryptocurrencies.

Although it is hard to decide whether a triplet is a DDH-triplet without knowing the multipliers, one could convince anyone in zero-knowledge that a tuple is indeed a DDH-tuple if one possesses the multipliers.

The language $\mathcal{L}_{DDH}$ is defined to be the set of all tuples (G, aG, bG, abG) where $G \in \mathbb{G}$ is of order prime q. The Chaum-Pedersen protocol enables a prover $\mathcal{P}$ to prove to a verifier $\mathcal{V}$ that $(G, A, B, C) \in \mathcal{L}_{DDH}$ in zero-knowledge for groups of prime order [9]. The protocol is organized as follows:

1. $\mathcal{V}$: $s \xleftarrow{\$} \mathbb{Z}_q$, then sends $commit(s)$
2. $\mathcal{P}$: $r \xleftarrow{\$} \mathbb{Z}_q$, then sends $y_1 = rG$, $y_2 = rB$.
3. $\mathcal{V}$ opens commitment by sending s
4. $\mathcal{P}$ sends $z = r + as \pmod q$
5. $\mathcal{V}$ checks $zG = y_1 + sA \pmod q \wedge zB = y_2 + sC \pmod q$

Note that in the following a non-interactive version of this protocol will only be considered that can be achieved by applying the Fiat-Shamir heuristic.

2.5 ECDSA with arbitrary generator element

Elliptic Curve Digital Signature Algorithm (ECDSA) is a key component of MixEth. ECDSA is widely deployed in practice, where in most cases signatures are generated and verified with respect to a fixed generator element of the underlying group [12]. Since all generators are equal from a security point of view, a single generator element is usually fixed in order to promote standardization and assist usability.

However, in MixEth, we deploy a somewhat loosened version of ECDSA, where we allow arbitrary generator elements to be used. Such an extension is indeed needed for withdrawing funds from the mixer, because shuffled public keys remain public keys with respect to non-standardized generator elements. Therefore the usual Sig and Vf algorithms for signing and verifying a messages gets an additional parameter G', which is not necessarily the standardized generator element. Key generation algorithm works as usual $(pk, sk) \xleftarrow{\$} \mathsf{KGen}(1^\lambda)$, on the other hand $\sigma \xleftarrow{\$} \mathsf{Sig}(G', sk, m)$ and $0/1 \leftarrow \mathsf{Vf}(G', pk, \sigma, m)$ accept new generators.

In our security proofs we will be relying on the fact that ECDSA is *existentially unforgeable* [12], i.e. no efficient adversary could forge a signature on any given message with non-negligible probability.

Note that although Ethereum does not support natively the verification of ECDSA signatures with respect to arbitrary generators, however it can easily be implemented in a smart contract.

2.6 Ethereum

Ethereum is a cryptocurrency built on top of a blockchain. Similarly to Bitcoin, network participants broadcast transactions in a peer-to-peer network, where transactions are bundled together into blocks that are appended to a public ledger called blockchain. Only those specific nodes can append new blocks to the blockchain who previously solved a difficult cryptographic puzzle. The state of the system consists of the state of different accounts populating it.

In Ethereum currently there are two types of accounts. The first account type is called *externally owned account*. It owns an ECDSA keypair controlled by its user. Private keys are used to sign transactions. On the other hand there are *contract accounts*, often smart contracts, that additionally have persistent storage and contract's code. Both of the account types have Ether balances, which is the native currency of the Ethereum network. Ether is denominated in wei, where $1\ ETH = 10^{18}\ wei$.

Transactions can alter the system's state by either creating a contract account or by calling to an existing account. Transactions to externally owned accounts can only transfere Ether, while transactions to contract accounts can additionally execute the code associated with them. Codes are executed in a quasi-Turing complete execution environment, called Ethereum Virtual Machine (EVM).

EVM is quasi-Turing complete, since smart contract's code cannot run indefinitely due to the so called *gas mechanism*. In every transaction the sender needs to pay upfront for the execution of the contract's code. The computational complexity of a transaction is measured in gas, which can be bought for Ether on a price set by the transaction originator, so called gas price. Therefore the transaction fee is the gas cost multiplied by the gas price. One needs to specify a gas limit, meaning that they do not allow their transaction to burn more gas than the limit. If a transaction during execution runs out of gas, then all the state changes are reverted, while the transaction fee is paid to the miner. If there is gas left after successful execution, transaction originator is reimbursed. Additionally there exists a block gas limit,

which limits the number of computational steps fitting in one block. Currently the block gas limit is cca. 8,000,000 gas. Naturally users of Ethereum are very much incentivized to minimize the gas cost of their transactions in order to spend as little as possible on transactions fees. Small gas costs are also crucial from a scalability point of view, since the less gas burnt for each transaction, the more transaction can fit into a single block.

2.6.1 State channels

Public blockchain's decentralization comes at an inherent cost in regard to scalability, since currently each full node verifies the full state of the public ledger. Often times Bitcoin's ($\sim$ 7 transactions per second (tps)) or Ethereum's ($\sim$ 15 tps) throughput is compared to that of Visa's ($\sim$ 45,000 tps). Blockchains' scalability issues became an increasingly growing problem as more and more users adopted the technology. One remarkable example was the launch of the Cryptokitties game in 2017, when the Ethereum network was congested for a few hours due to the enormous popularity of the game. Therefore several solutions were proposed to alleviate aforementioned scalability issues.

One of the major class of these techniques is called off-chain solutions. The insight of these proposals is that it is not needed to conduct all transactions on the blockchain, since participants could lock funds on-chain and afterwards securely issue transactions off-chain, for instance micropayments, with high degree of security and finality. Participants only need to get back on-chain if there is a dispute about what happened precisely off-chain or they would like to lock up funds and redeem them on-chain. The first implementation of this idea was a payment channel network for Bitcoin called Lightning network [24]. The advantage of the Lightning network is that participants can issue several payments without sending transactions to the blockchain and paying the sometimes costly transaction fees. Furthermore users are guaranteed to have instant finality instead of waiting several blocks to confirm their payments.

State channels are the more general form of payment channels, they can be used not only for payments, but for any arbitrary state updates on a blockchain, like changes inside a smart contract. State channels were first described in detail by Jeff Coleman et al.[10]. Since then several other frameworks for generalised state channels were elaborated [11, 15]. Recently a case study of the Battleship game was published by Patrick McCorry, Chris Buckland et al. to evaluate how state channels could contribute in scaling blockchain-based applications [15].

Later in this paper, in Section 6.2 we argue that MixEth can be made more scalable by implementing shuffling in a state channel.

3 Threat model

3.1 Participants and interactions

In a decentralized tumbler, we have 3 distinct entities: the tumbling smart contract, a set of senders and a set of receivers. A sender, whom we will call Alice, sends funds to the receiver, Bob, through the mixer contract in order to break direct links between their public keys. The list of contract identifiers associated with distinct sessions is denoted as $tumblers$. In all the following interactions and algorithms we assume that the public state of the tumbler is implicitly given as input. Interactions of these entities can be summarized as follows:

$tx \xleftarrow{\$} Deposit(tumblers, sk_A, pk_B)$: The sender runs this algorithm to deposit a predefined amount of ether to the receiver's public key.

$0/1 \leftarrow VerifyDeposit(tx)$: The tumbler contract checks the validity of senders' deposits.

$ProcessDeposit(tx)$: upon receiving a valid deposit transaction, the mixing contract updates its internal state accordingly.

Let PK_0 denote the set of public keys to be mixed after the depositing period. Generally PK_i will denote the set of shuffled public keys after i shuffling round. Furthermore let us set $C_0^* = G$, the standard generator point of the secp256k1 curve. Similarly C_i^* denotes the shuffling accumulated constant after the ith shuffling round. Shuffles are computed off-chain, the outputs of the algorithm are written into the on-chain mixer contract. Anyone is allowed to shuffle the public keys by paying some deposit to the tumbler contract.

$PK_{i+1}, C_{i+1}^*, proof_{DDH}(G, c_iG, C_i^*, C_{i+1}^*) \xleftarrow{\$} Shuffle(PK_i, C_i^*, c_i, \pi_i)$. The new shuffling accumulated constant C_{i+1}^* can be obtained by $C_{i+1}^* = c_iC_i^*$. The shuffling accumulated constant is needed for receivers to audit shuffling and to collect their funds at the end of the final shuffling period. The permutation π_i and the secret multiplier c_i from the new shuffling accumulated constant should be kept private after shuffling, otherwise it is trivial to track how public keys are shuffled. All the outputs of the $Shuffle$ algorithm are public and written into the tumbling contract.

Note that we need an additional Chaum-Pedersen proof from the shuffler in order to prove that the shuffler knows a secret multiplier between the new and the previous shuffling accumulated constants. If we did not require such proof, a malicious shuffler could break anonymity just by uploading a random shuffle of the original public keys, the set PK_0.

$0 \vee 1 \leftarrow ProcessShuffle(PK_{i+1}, C_{i+1}^*, proof_{DDH}(G, c_iG, C_i^*, C_{i+1}^*))$. If the mixing contract is in a shuffling period and the Chaum-Pedersen proof is verified, then (PK_{i+1}, C_{i+1}) is written into the contract state, otherwise shuffling transactions is reverted.

$0 \vee 1 \leftarrow ChallengeShuffle(PK_i, C_i^*, PK_{i-1}, C_{i-1}^*, pk_B)$: receiver B with public key $pk_B = s_BG$ can challenge an incorrect shuffle at the ith round by giving a Chaum-Pedersen zero-knowledge proof that the following tuple is DDH-tuples: $(C_{i-1}^*, s_BC_{i-1}^*, C_i^*, s_BC_i^*)$. If the proof verifies and $s_BC_i^* \notin PK_i$, while $s_BC_{i-1}^* \in PK_{i-1}$, then the challenge is accepted, otherwise rejected. This proof and checks allow one to be certain that indeed the ith round is the first round in which the corresponding public key to s_B is shuffled incorrectly.

$tx \xleftarrow{\$} WithdrawShufflingDeposit(sk_B)$: after a challenging period a shuffler can withdraw their shuffling deposit from the tumbler contract.

$0 \vee 1 \leftarrow VerifyWitdhrawShufflingDeposit(pk_B)$: if there was no successful challenges against the shuffler, i.e. their deposit is not slashed, they can withdraw their shuffling deposit.

$tx \xleftarrow{\$} Withdraw(sk_B, C_{final}^*)$: after the end of the shuffling period users are allowed to withdraw their funds. Note that here withdraw transactions will be signed with a modified version of ECDSA, where not the original generator element G is used as generator rather C_{final}^*, the final shuffling accumulated constant.

$0/1 \leftarrow VerifyWithdraw(tx)$: tumbler checks the validity of a recipient's withdrawal transaction.

$ProcessWithdraw(tx)$: upon receiving a valid withdrawal transaction, mixing contract updates its internal state accordingly.

3.2 Security goals

We are aiming to achieve and prove the same notions of security as the ones defined in [16], namely anonymity, availability and theft prevention. These notions of anonymity, availability and theft prevention were introduced in [16], which are included in the Appendices for sake of self-containedness.

We are going to assume that at most $n-2$ recipients are malicious (n is the number of recipients). Otherwise, no meaningful notion of security can be achieved. Furthermore we presume that participants are on-line during the entire course of mixing in order to be able to monitor and potentially challenge any incorrect shuffle. Finally we assume that honest recipients will always exercise their rights to shuffle and they do not disclose any private information used in their shuffles.

Hereby we only give intuition for the notions of security, for formal definitions the astute reader is referred to the Appendices.

3.2.1 Anonymity

Sender anonymity is achieved if an adversary cannot determine to whom honest senders are sending funds, assuming that honest senders' deposits are indistinguishable.

Recipient anonymity is achieved if honest recipients withdrawal transactions are indistinguishable.

3.2.2 Availability

It is essential for a coin mixer to provide availability, meaning that honest recipients can always withdraw their money from the mixer, even if senders and all but one recipients are compromised.

Adversary $\mathcal{A}$ wins the availability security game if they manage to get the tumbler into a state where honest recipient cannot withdraw their funds.

3.2.3 Theft prevention

We would like to ensure that neither coins can be withdrawn twice, nor withdrawn by anyone other but the intended recipient.

4 MixEth

MixEth is a coin mixing smart contract allowing parties to efficiently tumble coins in a trustless manner on Ethereum.

4.1 Initializing the tumbler and depositing period

A MixEth contract living on the Ethereum blockchain at $id_{contract}$ address must be initialized with the amt parameter, which denotes the denomination of ether to be mixed. Every sender must deposit exactly amt ether to a specific public key. Deposits with incorrect ether value or invalid public key are rejected. Public keys in subsequent deposit transactions are written into the $initPubKeys[]$ array.

4.2 Shuffling period

After the depositing round, shuffling and challenging rounds are coming after in turns. Each shuffling round is followed by a challenging round when the correctness of the preceding shuffle can be challenged by anyone. If a challenge is accepted, then shuffler's deposit is lost and given to the challenger, the incorrect shuffle is discarded and shuffling continues from the set of public keys prior to the discarded shuffle. In the course of a shuffle an honest shuffler should multiply all the public keys with a secret multiplier c and then permute all the

transformed public keys. Honest shuffler commits to c by sending back to MixEth the new shuffling accumulated constant and the shuffled public keys along with a Chaum-Pedersen proof, proving the correctness of the new shuffling accumulated constant.

Computing the shuffle is done off-chain, see Procedure 1, however the new set of shuffled public keys, the updated shuffling accumulated constant and the Chaum-Pedersen proof are loaded into the MixEth contract enabling anyone to verify the shuffle's correctness and to continue public key shuffling after the corresponding challenging round. In Procedure 1 the function $generateChaumPedersen(G, A, B, C)$ denotes a PPT algorithm, which generates a Chaum-Pedersen proof, proving that $\log_G(A) = \log_B(C)$. The mixing contract accepts a shuffling transaction if and only if the contract is in a shuffling period and the Chaum-Pedersen proof is verified, otherwise rejects.

Algorithm 1 Off-chain public key shuffling algorithm for the ith shuffling round.

$PK_i \leftarrow []$
$c \xleftarrow{\$} \mathbb{Z}_n$
$C^*_{i-1} \leftarrow read\ from\ MixEth\ contract$
$PK_{i-1} \leftarrow read\ from\ MixEth\ contract\ the\ current\ sequence\ of\ shuffled\ public\ keys$
$\pi \xleftarrow{\$} S_{|PK_{i-1}|}$
for $j = 0; j < |PK_{i-1}|; j++$ **do**
    $PK_i[\pi(j)] = c * PK_{i-1}[j]$
end for
$C^*_i = cC^*_{i-1}$
$proof_{DDH} = generateChaumPedersen(G, cG, C^*_{i-1}, C^*_i)$
Output: $(PK_i, C^*_i, proof_{DDH})$

4.3 Challenging period

Every participant should check the correctness of incoming shuffles, therefore sufficient time should be provided for each challenging round. These are the actions Bob as a receiver needs to perform to check the correctness of the shuffle at ith round if Bob has secret key s_B. In this case Bob should check whether $s_BC^*_i \in PK_i$ or not. If not, Bob should prove to MixEth that the ith round is indeed the first round, where the shuffled public key corresponding to s_B is compromised. The Chaum-Pedersen proof in the challenge transaction ensures that the integrity of the shuffled public key in round $i-1$st is intact, while shuffled public key is compromised in the ith round.

Algorithm 2 On-chain verification algorithm of incoming shuffle challenges.

Input$(PK_i, PK_{i-1}, proof_{DDH}(C^*_{i-1}, s_BC^*_{i-1}, C^*_i, s_BC^*_i)$
$b \leftarrow verifyChaumPedersen(proof_{DDH}(C^*_{i-1}, s_BC^*_{i-1}, C^*_i, s_BC^*_i))$
$b^* \leftarrow 0$
if $b \wedge s_BC^*_{i-1} \in PK_{i-1} \wedge s_BC^*_i \notin PK_i$ **then**
    $b^* \leftarrow 1$
else
    $b^* \leftarrow 0$
end if **Output:** b^*

Note that every recipient should perform this check after each shuffling. Noone can check the inclusion and correctness of shuffled public keys for recipients other than themselves. This task is non-outsourcable unless one reveals her own private key, which would obviously lead to loss of funds at the end of the MixEth protocol, since anyone can claim the funds knowing the corresponding secret key.

In Procedure 2 $verifyChaumPedersen(proof_{DDH})$ denotes a deterministic polynomial-time algorithm which verifies the correctness of a Chaum-Pedersen zero-knowledge proof. The algorithm outputs 1 if the proof is verified, otherwise 0.

4.4 Withdrawing

Let C^*_{final} be the final shuffling accumulated constant. For a recipient B, whose public key $s_BG \in initPubKeys[]$, in the final shuffle there will be $s_BC^*_{final}$. The recipient can prove to MixEth that she knows secret key s_B by signing their public key using a modified ECDSA, which uses C^*_{final} as the generator element instead of the standardized G.

5 Security

Notions of security are proven in the Appendices.

6 Implementation

We implemented MixEth with two different approaches. The first implementation of MixEth does not apply state channels, all the transactions are made on-chain. This could lead to unwanted gas costs as the number of corrupted shuffles increases. One of our main motivation with MixEth is to provide an efficient and scalable coin mixing protocol which uses as little blockchain resources, storage and gas, as possible. Therefore we also implement and evaluate MixEth applying state channels, namely shuffling and challenging a shuffle occurs off-chain and only deposit and withdrawal transactions happen on-chain. Both of the implementations allow users to mix Ether or other ERC20-compatible, a popular Ethereum token standard, tokens.

One of the main bottlenecks of coin mixing protocols is the withdrawal transactions' gas costs. A Miximus withdrawal transaction burns 1,903,305 gas, regardless of the number of participating parties. Since the block gas limit is 8,000,266 as of 2018, October 24 only 4 Miximus withdrawal transactions could fit in one Ethereum block. This is even worse for Möbius, since the gas cost for withdrawing coins from a Möbius mixer contract linearly increases with the numbers of participants.

Although MixEth is more gas-efficient than Möbius or Miximus, it incurs a higher time-complexity, ie. recipients need to expect longer delays for funds to arrive since each challenging period lasts a few blocks of time. Furthermore MixEth requires users to be online during the course of mixing, in some scenarios this might be a demanding requirement.

All MixEth smart contracts were written in the Solidity language, which is currently the dominant language for developing Ethereum smart contracts. All MixEth contracts are available online[3].

[3] `https://github.com/seresistvanandras/MixEth`

6.1 Fully on-chain implementation

Conceivably mixers would like to minimize off-chain coordination, therefore in our first implementation of the MixEth protocol, we assumed that all transactions will take place on-chain. There is only a single off-chain message from receiver to sender, where receiver delivers their public key to the sender. The rest of the protocol happens entirely on-chain.

On-chain storage is extremely expensive: it requires 20,000 gas to store a 256-bit number, however if a particular storage slot is already taken and one wants to overwrite it with a non-zero element then storing only consumes 5000 gas. To minimize on-chain storage costs, only the last two list of shuffled public keys are stored in the MixEth contract's permanent storage. Note that storing only the latest list of shuffled public keys would not be enough, since honest receivers could not prove to the contract that their shuffled public key is compromised unless also the last but first list of shuffled keys is also available for the contract to check the Chaum-Pedersen proof against. Such a storage structure implies that after uploading the new list of shuffled public keys, a challenging period should proceed in order to let receivers check the correctness of the shuffle and whether their shuffled public key is stored in the smart contract. Furthermore we also allow senders to shuffle and deposit new public keys at the same time, meaning that only 3 on-chain transactions (shuffle, withdraw mixed coins and withdraw shuffling deposit) are sufficient to complete the protocol.

A great advantage of the fully on-chain version of MixEth is that it allows dynamic anonymity sets. One could potentially deposit funds to the contract and shuffle public keys and leave funds in the mixing contract for indefinite amount of time. As soon as the anonymity set is large enough a receiver could withdraw their assets. A receiver in a MixEth contract with N senders could withdraw their funds after $N^{'}$ shuffling rounds, where $N^{'}$ is arbitrary. This dynamic nature of the contract could even lead to a single monolithic MixEth contract instead of having multiple MixEth contracts with significantly fragmented anonymity sets. A single MixEth contract is able to support the mixing of ether and ERC-20 compatible tokens as well. However note that the gas complexity of shuffling transactions grows linearly in the number of participants, therefore the fully on-chain implementation is not capable to support extremely large anonymity sets with participants more than a few hundreds.

6.2 State channel implementation

We have also adapted MixEth to operate within a state channel. We wrote the implementation within the guidelines of the Counterfactual framework [10]. This allowed us to delegate the processes of setup, liveness disputes and finalisation to the framework so that we could focus on adapting the application logic. Unlike the on-chain implementation the state channel implementation requires that the set of participants be agreed upon upfront. In state channels each update to the state needs to be signed by all other participants, this means that state channel applications are inherently at least $\mathcal{O}(n)$. To co-ordinate these off-chain updates the Counterfactual framework enforces that all applications be turn based, introducing a turn taker for each turn who may propose a new state. The original MixEth implementation was not turn based so we have adapted the application to this constraint, an example of this adaptation is the challenge round. In the on-chain implementation a time period is allowed during which any participant may challenge, we have adapted this by proceeding turn-based through the participants offering each the chance to either challenge or pass. In the case of a breakdown in cooperation in the channel, a liveness fault, it has been shown that all operations succeeding the cooperation breakdown must proceed on chain[15] or be abandoned at some financial cost specified by the application, meaning that if every

shuffle were to be succeeded by a challenge round each participant would be forced, by threat this lost deposit, to make an on-chain transaction after each shuffle, incurring $\mathcal{O}(n)$ on chain operations. To mitigate this we removed the challenge after each round and instead introduced a challenge round that takes place after all shuffles have completed, during this round any of the preceding shuffles may be challenged.

Given these adaptations the application proceeds as follows, all participants including senders, shufflers and receivers, deposit funds in a mutli-signature wallet compatible with the Counterfactual framework, they then follow the installation protocols specified by the framework to install the adapted MixEth logic. Afterwards each participant signs a transaction that transfers an equal amount to each withdrawer from the multi-sig, dependent on correct execution of the channelised MixEth application logic. This application logic proceeds as follows: each sender names a public key of a shuffler as in the deposit stage of the on-chain application, then each shuffler takes it in turn to shuffle. After all shuffles have taken place each withdrawer is given a turn to either declare fraud or no-fraud on any shuffle round. Finally each withdrawer then provides proof of ownership by submitting a valid signature on the modified ECDSA scheme. If any of these steps does not occur, or does not occur correctly, the protocol aborts and the conditional transfer does not occur. In this case the perpetrator loses a deposit, either through fraud proof or through failure to take their turn when state is published on-chain. A further modification would be to distribute the slashed deposit to each of the other participants, compensating them for their lost time and the gas costs associated with proving the fault of the other party. Following this protocol the on-chain transactions are now reduced to: one transaction from each participant to deposit funds into the multi-sig, and a set of transactions that send funds from the multi-sig to each of the withdrawers and deposits back to each of the other participants.

Table 1 Proof-of-concept implementation gas cost results. Expect further improvements. MixEthChannel refers to the implementation which leverages state channels for shuffling and challenging periods.

	Deployment	Deposit	Shuffle		Withdraw
			Shuffle upload	Challenge	
Möbius [16]	1,046,027	76,123	0	0	335,714n
Miximus [1]	1,751,378	732,815	0	0	1,903,305
MixEth	5,395,945	99,254	$366{,}216 + 10{,}000n$	227,563	113,265
MixEthChannel	672,276	21,000	0	0	26,749

7 Related work

Möbius was the very first trustless coin mixer designed for Ethereum[16]. Authors of Möbius provided formal definitions of various notions of security such as anonymity, theft prevention and mixer availability. These properties could be used to evaluate and compare existing and future proposals from a security perspective. Möbius is a ring-signature-based trustless coin mixer with minimal on-chain transaction complexity: users of Möbius just need to create a deposit and a withdraw transaction. However the gas cost of the withdrawal transaction increases linearly in the number of receivers, which limits the size of possible anonymity sets. No more than 24 reciever could use Möbius with current cca.8,000,000 block gas limit. If more people tried to use the mixer funds would be stucked in the mixer contract, since the gas costs of withdrawal transactions would be greater than the block gas limit.

Miximus is a zkSNARK-based mixer for Ethereum[1]. It uses zkSNARKs to conceal the mapping between depositors and recipients. A depositor creates a leaf in a Merkle-tree. A depositor needs to exchange the preimage of the leaf with the recipient. Later, a recipient could prove to the Miximus contract that they know one of the preimages of a certain, undisclosed leaf. So called nullifiers enable recipients to withdraw funds once and only once. The gas costs of depositing and withdrawing funds from a Miximus mixer is independent of the number of participants. However there are disadvantages of this approach; Miximus only provides anonymity against outsiders, since if Alice funds to Bob via Miximus, Alice will know when Bob made the withdrawal transaction. Another, more severe limitation of Miximus is the trusted setup required for generating the proving key for the zkSNARK. If this trusted setup is compromised, the deployer of the contract, who generated the proving key could potentially steal funds from the mixer. Although, this issue could be amended somehow via a multi-party computation (MPC) further increasing the off-chain communication complexitiy of Miximus.

As Table 2 demonstrates, both Möbius and Miximus require 2 on-chain transactions, while MixEth requires 3. In spite of this seemingly added complexity, the 3 on-chain transactions to complete the MixEth protocol (deposit, shuffle, withdraw) consume significantly less gas than those (deposit, verify linkable ring signature/zkSNARK) of Möbius and Miximus, see Table 1.

Table 2 Number of on-chain transactions and off-chain messages per a single participant required to run a certain coin mixer protocol. Note that in case of Miximus if one wants to avoid the trusted setup for the zkSNARK, then they need to perform a secure multi-party computation protocol to trust-minimize the proving key generation.

	#Off-chain messages	#Transactions
Centralized		
Mixcoin [6]	2	2
Blindcoin [26]	4	2
TumbleBit [13]	12	4
Decentralized		
Coinjoin [14]	$\mathcal{O}(n^2)$	1
Coinshuffle [25]	$\mathcal{O}(n)$	1
XIM [4]	0	7
Möbius [16]	2	2
Miximus [1]	1+MPC	2
MixEth	1	3
MixEthChannel	$\mathcal{O}(n)$	2

References

1 barryWhiteHat. Miximus. `https://github.com/barryWhiteHat/miximus`, 2018.

2 barryWhiteHat. Miximus gas costs. `https://www.reddit.com/r/ethereum/comments/8ss53z/miximus_zksnark_based_anonymous_transactions_is/`, 2018.

3 Stephanie Bayer and Jens Groth. Efficient zero-knowledge argument for correctness of a shuffle. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 263–280. Springer, 2012.

4 George Bissias, A Pinar Ozisik, Brian N Levine, and Marc Liberatore. Sybil-resistant mixing for bitcoin. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, pages 149–158. ACM, 2014.

5 Dan Boneh. The decision diffie-hellman problem. In *International Algorithmic Number Theory Symposium*, pages 48–63. Springer, 1998.

6 Joseph Bonneau, Arvind Narayanan, Andrew Miller, Jeremy Clark, Joshua A Kroll, and Edward W Felten. Mixcoin: Anonymity for Bitcoin with accountable mixes. In *International Conference on Financial Cryptography and Data Security*, pages 486–504. Springer, 2014.

7 Vitalik Buterin and Nick Johnson. EIP86: Account Abstraction. `https://github.com/ethereum/EIPs/blob/master/EIPS/eip-86.md`, 2017.

8 Wren Chan and Aspen Olmsted. Ethereum transaction graph analysis. In *2017 12th International Conference for Internet Technology and Secured Transactions (ICITST)*, pages 498–500. IEEE, 2017.

9 David Chaum and Torben Pryds Pedersen. Wallet databases with observers. In *Annual International Cryptology Conference*, pages 89–105. Springer, 1992.

10 Jeff Coleman, Liam Horne, and Li Xuanji. Counterfactual: Generalized state channels, 2018.

11 Stefan Dziembowski, Lisa Eckey, Sebastian Faust, and Daniel Malinowski. Perun: Virtual payment channels over cryptographic currencies. Technical report, IACR Cryptology ePrint Archive, 2017: 635, 2017.

12 Manuel Fersch, Eike Kiltz, and Bertram Poettering. On the provable security of (EC) DSA signatures. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1651–1662. ACM, 2016.

13 Ethan Heilman, Leen Alshenibr, Foteini Baldimtsi, Alessandra Scafuro, and Sharon Goldberg. TumbleBit: An untrusted Bitcoin-compatible anonymous payment hub. In *Network and Distributed System Security Symposium*, 2017.

14 Greg Maxwell. CoinJoin: Bitcoin privacy for the real world. In *Post on Bitcoin forum*, 2013.

15 Patrick McCorry, Chris Buckland, Surya Bakshi, Karl Wüst, and Andrew Miller. You sank my battleship! A case study to evaluate state channels as a scaling solution for cryptocurrencies.

16 Sarah Meiklejohn and Rebekah Mercer. Möbius: Trustless tumbling for transaction privacy. *Proceedings on Privacy Enhancing Technologies*, 2018(2):105–121, 2018.

17 Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M Voelker, and Stefan Savage. A fistful of bitcoins: characterizing payments among men with no names. In *Proceedings of the 2013 conference on Internet measurement conference*, pages 127–140. ACM, 2013.

18 Pedro Moreno-Sanchez, Muhammad Bilal Zafar, and Aniket Kate. Listening to whispers of ripple: Linking wallets and deanonymizing transactions in the ripple network. *Proceedings on Privacy Enhancing Technologies*, 2016(4):436–453, 2016.

19 Malte Moser, Rainer Bohme, and Dominic Breuker. An inquiry into money laundering tools in the Bitcoin ecosystem. In *eCrime Researchers Summit (eCRS), 2013*, pages 1–14. IEEE, 2013.

20 Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

21 Nchinda Nchinda. Exploring Pseudonimity on Ethereum. `https://media.consensys.net/exploring-pseudonimity-on-ethereum-dda257019eb4`, 2016.

22 Serge Nedashkovsky. Huge Ethereum Mixer. `https://blog.cyber.fund/huge-ethereum-mixer-6cf98680ee6c`, 2017.

23 C Andrew Neff. A verifiable secret shuffle and its application to e-voting. In *Proceedings of the 8th ACM conference on Computer and Communications Security*, pages 116–125. ACM, 2001.

24 Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments. `https://lightning.network/lightning-network-paper.pdf`, 2016.

25 Tim Ruffing, Pedro Moreno-Sanchez, and Aniket Kate. CoinShuffle: Practical decentralized coin mixing for Bitcoin. In *European Symposium on Research in Computer Security*, pages 345–364. Springer, 2014.

26 Luke Valenta and Brendan Rowan. Blindcoin: Blinded, accountable mixes for bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 112–126. Springer, 2015.

27 Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151:1–32, 2014.

A Formal definitions of security

Hereby we formally define the notions of security MixEth is aiming to achieve.

In the security definitions and games introduced by [16] adversary $\mathcal{A}$ might have access to the following oracles. CORR enables $\mathcal{A}$ to corrupt a sender or receiver by learning the secret key of any party l of their choice. Oracle access to AD or AW allow $\mathcal{A}$ to deposit or withdraw respectively from tumbler session j. Furthermore $\mathcal{A}$ might instruct honest senders or receivers to deposit or withdraw from tumbler session j by using oracles HD and HW. In the following C denotes the set of corrupted parties, while the list of honest deposits and withdrawals are denotes as H_d and H_w respectively.

These oracles are formally defined as follows:

AD(tx,j)

$b \leftarrow VerifyDeposit(tumblers[j], tx)$
$if\ (b)\ ProcessDeposit(tumblers[j], tx)$
return b

AW(tx,j)

$b \leftarrow VerifyWithdraw(tumblers[j], tx)$
$if\ (b)\ ProcessWithdraw(tumblers[j], tx)$
return b

CORR(l)

$C = C.push(pk_{B_l})$
return sk_{B_l}

HD(i,j,l)

$tx \xleftarrow{\$} Deposit(sk_{A_i}, pk_{B_l})$
$H_d = H_d.push(tx)$
$ProcessDeposit(tumblers[j], tx)$
return tx

HW(j,l)

$if\ (pk_{B_l} \notin tumblers[j].keys_B)\ return\ \perp$
$tx \xleftarrow{\$} Deposit(sk_{A_i}, pk_{B_l})$
$H_w = H_w.push(j, l, tx)$
$ProcessWithdraw(tumblers[j], tx)$
return tx

A.1 Anonymity

▶ **Definition 1.** *Define* $\boldsymbol{Adv}^{d-anon}_{mix,\mathcal{A}}(\lambda) = 2\Pr[G^{d-anon}_{mix,\mathcal{A}}(\lambda)] - 1$ *for* $d \in \{dep, with\}$*, where these games are defined as follows:*

MAIN $\mathsf{G}^{dep-anon}_{mix,\mathcal{A}}(\lambda)$

$(pk_i, sk_i) \xleftarrow{\$} \mathsf{KGen}(1^\lambda)\ \forall i \in [n]$
$\mathsf{PK}_A \leftarrow \{pk_i\}_{i=1}^n; C, H_d, tumblers \leftarrow \emptyset$
$b \xleftarrow{\$} \{0,1\}$
$(state, j, pk, i_0, i_1) \xleftarrow{\$} \mathcal{A}^{CORR,AD,HD,AW}(1^\lambda, \mathsf{PK}_A)$
$tx \xleftarrow{\$} Deposit(tumblers[j], sk_{A_{l_b}}, pk)$
$b' \xleftarrow{\$} \mathcal{A}^{CORR,AD,HD,AW}(state, tx)$
return $b = b'$

MAIN $\mathsf{G}^{with-anon}_{mix,\mathcal{A}}(\lambda)$

$(pk_i, sk_i) \xleftarrow{\$} \mathsf{KGen}(1^\lambda)\ \forall i \in [n]$
$\mathsf{PK}_B \leftarrow \{pk_i\}_{i=1}^n; C, H_d, tumblers \leftarrow \emptyset$
$b \xleftarrow{\$} \{0,1\}$
$(state, j, pk, l_0, l_1) \xleftarrow{\$} \mathcal{A}^{CORR,AD,HD,AW}(1^\lambda, \mathsf{PK}_B)$
$\mathsf{PK} \leftarrow tumblers[j].keys_B$
$if(pk_{B_{l_0}} \notin \mathsf{PK}) \vee (pk_{B_{l_1}} \notin \mathsf{PK})\ return\ 0$
$tx \xleftarrow{\$} Withdraw(tumblers[j], sk_{B_{l_b}})$
$b' \xleftarrow{\$} \mathcal{A}^{CORR,AD,HD,AW}(state, tx)$
$if(pk_{l_b} \in C\ for\ b \in \{0,1\})\ return\ 0$
$if((j, l_b, \cdot)) \in H_w\ for\ b \in \{0,1\})\ return\ 0$
return $b = b'$

Then the tumbler satisfies sender or recipient anonymity if for all PPT adversaries $\mathcal{A}$ *there exists a negligible function* $\nu(\cdot)$ *such that* $\boldsymbol{Adv}^{dep-anon}_{mix,\mathcal{A}}(\lambda) < \nu(\lambda)$ *or* $\boldsymbol{Adv}^{with-anon}_{mix,\mathcal{A}}(\lambda) < \nu(\lambda)$ *respectively.*

A.2 Availability

▶ **Definition 2.** *Define* $\boldsymbol{Adv}^{avail}_{mix,\mathcal{A}}(\lambda) = \Pr[G^{avail}_{mix,\mathcal{A}}(\lambda)]$*, where the game is defined as follows:*

MAIN $\mathsf{G}^{avail}_{mix,\mathcal{A}}(\lambda)$

$(pk_i, sk_i) \xleftarrow{\$} \mathsf{KGen}(1^\lambda)\ \forall i \in [n]$
$\mathsf{PK}_B \leftarrow \{pk_i\}_{i=1}^n; C, H_w \leftarrow \emptyset$
$(l, j) \xleftarrow{\$} \mathcal{A}^{CORR,AD,HW,AW}(1^\lambda, \mathsf{PK}_B)$
$b \leftarrow \mathsf{VerifyWithdraw}(\mathsf{tumblers}[j], \mathsf{Withdraw}(sk_l))$
$if((pk_l \in C)\ \vee\ ((j, l, \cdot) \in H_w))\ return\ 0$
return $(b = 0) \wedge (pk_l \in tumblers[j].keys_B)$

Then the tumbler satisfies availability if for all PPT adversaries $\mathcal{A}$ *there exists a negligible function* $\nu(\cdot)$ *such that* $\boldsymbol{Adv}^{avail}_{mix,\mathcal{A}}(\lambda) < \nu(\lambda)$*.*

A.3 Theft prevention

▶ **Definition 3.** *Define* $\boldsymbol{Adv}^{theft}_{mix,\mathcal{A}}(\lambda) = \Pr[G^{theft}_{mix,\mathcal{A}}(\lambda)]$*, where the game is defined as follows:*

$\underline{\text{MAIN } \mathsf{G}^{theft}_{mix,\mathcal{A}}(\lambda)}$

$(pk_i, sk_i) \xleftarrow{\$} \mathsf{KGen}(1^\lambda)\ \forall i \in [n]$

$\mathsf{PK}_B \leftarrow \{pk_i\}_{i=1}^n; C, H_w, \mathsf{contract} \leftarrow \emptyset$

$(\mathsf{tx}, j) \xleftarrow{\$} \mathcal{A}^{CORR,AD,AW,HW}(1^\lambda, \mathsf{PK}_B)$

$if(\mathsf{tumblers}[j].\mathsf{keys}_B \not\subset \mathsf{PK}_B \setminus C)$ *return* 0

return $\mathsf{VerifyWithdraw}(\mathsf{tumblers}[j], \mathsf{tx})$

Then the tumbler satisfies theft prevention if for all PPT adversaries $\mathcal{A}$ *there exists a negligible function* $\nu(\cdot)$ *such that* $\boldsymbol{Adv}^{theft}_{mix,\mathcal{A}}(\lambda) < \nu(\lambda)$*.*

B Proofs of security

This section provides informal ideas to the security proofs for the notions of security introduced formally above and informally in Section 3.2.

B.1 Recipient anonymity

The withdrawing transaction for recipient B sends funds to the public key $s_B C^*$. This public key does not reveal any links to the original $s_B G$ in case if at least one honest sender shuffled and the DDH assumption holds. Adversary can only distinguish between honest recipients public keys with negligible probability. See reduction proof in Appendix C.

B.2 Availability

If an adversary is able to destroy an honest recipient's funds' availability, it implies that adversary $\mathcal{A}$ either breaks the completeness of the Chaum-Pedersen protocol or successfully launched an eclipse attack against the honest recipient, who cannot send any transactions to honest Ethereum peers.

B.3 Theft prevention

If an adversary is able to steal funds from other users than it would imply that they managed to create a valid message/signature, (m, σ) pair for the final shuffled public key of an honest recipient without having access to the secret key of the honest recipient. This contradicts to the assumption that ECDSA is existentially unforgeable. Reduction proof is enclosed in Appendix E.

C Proof of Anonymity

Hereby we show that if there exists an adversary $\mathcal{A}$ who is able to break withdrawal anonymity defined in Section 3.2.1, then there exists another adversary $\mathcal{B}$ who is able to break the DDH assumption.

Towards contradiction let us assume that the recipient anonymity does not hold. Let us assume that the challenge to the DDH-adversary $\mathcal{B}$ is of the form $(sG, cG, c_0 G)$. In a

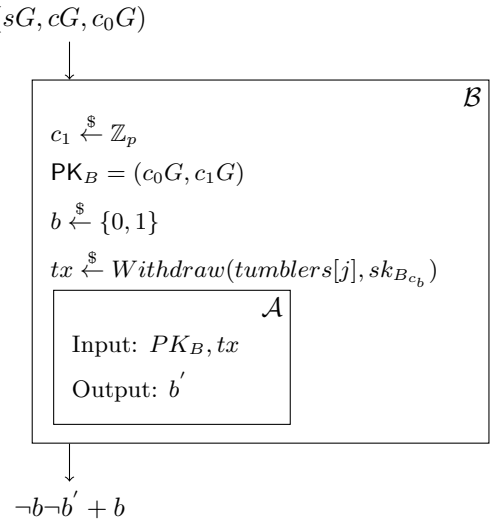


Figure 1 An illustration for the reduction of withdrawal anonymity to the DDH assumption.

DDH-game the adversary's goal is to decide whether c_0G is a random group element or it equals to scG. At the end of the DDH-game adversary outputs 1 if $c_0G = scG$ and 0 otherwise. Adversary $\mathcal{B}$ generates uniformly random public key c_1G and invokes $\mathcal{A}$ with the set $\mathsf{PK}_B = (c_0C^*, c_1C^*)$, then $\mathcal{B}$ forwards PK_B to $\mathcal{A}$. Then a withdrawal transaction occurs from c_bG. After polynomial-time $\mathcal{A}$ outputs b' and $\mathcal{B}$ will output $\neg b'$. If $\mathcal{A}$ outputs 0 and $b = b'$, this signals to $\mathcal{B}$ that c_0G might potentially be of the form $scG = c_0G$ i.e. it is a DDH tuple, therefore $\mathcal{B}$ outputs 1. In all the other cases $\mathcal{B}$ outputs a random bit. Therefore we have that $\mathcal{B}$ has an adventage in their DDH-game if and only if $\mathcal{A}$ wins their $\mathsf{G}^{with,anon}_{mix,\mathcal{A}}$ security game. Since we assumed that recipient anonymity does not hold we have that

$$\Pr[\mathsf{DDH}_{\mathcal{B}}] = \frac{1}{2} + \frac{1}{2} * \Pr[\mathsf{G}^{with,anon}_{mix,\mathcal{A}}] = \frac{1}{2} + \frac{1}{2} * \mathsf{non\text{-}negl}(\lambda),$$

which contradicts to the DDH assumption.

D Proof of Availability

The only possibility for an adversary to threaten the availability of funds for an honest receiver if they create an incorrect shuffle, where honest receiver's shuffled public key is compromised. Since the Chaum-Pedersen zero-knowledge protocol is complete, an honest receiver is alway able to create a Chaum-Pedersen proof, which demonstrates to the contract that their shuffled public key is compromised. Therefore we have for any PPT $\mathcal{A}$ and $\forall\lambda \in \mathbb{N}$ that,

$$\Pr[\mathsf{G}^{avail}_{mix,\mathcal{A}}] = 0 < negl(\lambda).$$

E Proof of Theft Prevention

Towards contradiction we assume that there exists an adversary $\mathcal{A}$, who is able to break the theft prevention property introduced in Section 3.2.3 with non-negligible probability. Using such an adversary as a subroutine we could create another efficient adversary $\mathcal{B}$ who is able to break the existential unforgeability of ECDSA. The input of the forgeability game is the security parameter which is forwarded to $\mathcal{A}$ along with n randomly generated public keys. By assumption $\mathcal{A}$ outputs with non-negligible probability valid withdraw transaction

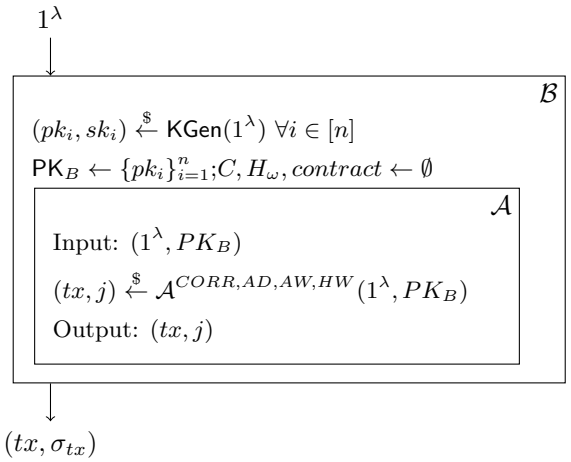


Figure 2 An illustration for the reduction of theft prevention to the existentially unforgeability of ECDSA.

belonging to one of the public keys in the mixer. A valid withdraw transaction is a (tx, σ_{tx}) pair, where σ_{tx} is a valid signature on transaction tx. Adversary $\mathcal{B}$ outputs the withdraw transaction and the ECDSA signature on it. $\mathcal{B}$ wins the forgeability game if and only if $\mathcal{A}$ wins their $\mathsf{G}^{theft}_{mix,\mathcal{A}}$ game:

$$\Pr[\mathsf{Forge}_{ECDSA,\mathcal{B}}] = \Pr[\mathsf{G}^{theft}_{mix,\mathcal{A}}] = \frac{1}{\lambda^\alpha},$$

for some fixed α. This contradicts to the assumption that ECDSA is existentially unforgeable.

F Extensions and improvements

MixEth is not fully compatible with the current EVM, however it could be deployed with a workaround. A recipient could ask another party or service to send a signed transaction including a signature which uses the modified version of ECDSA, where the generator element is the shuffling accumulated constant. MixEth could check this signature and send out funds to a fresh Ethereum address given in the withdraw transaction.

In the current design of MixEth if sender, Alice and receiver, Bob would like to use the mixer several times, Bob needs to share his receiver address in a secure communication channel with Alice as many times as he would like to receive payments. This communication overhead could be overcome by applying stealth addresses, where Bob needs to share once his public master key with Alice in order to receive arbitrary number of payments from her.

F.0.1 Ethereum account abstraction

Unfortunately, neither Möbius nor Miximus can be deployed on the present-day Ethereum. When users of the coin mixing contract, either Möbius or Miximus would like to withdraw their funds they cannot do this from a fresh address, since it does not hold any ether. Since as of now only the sender of a transaction can pay for the gas fee, users cannot withdraw their funds unless they ask someone to fund their fresh address.

Another solution for this problem is the Ethereum Improvement Proposal (EIP) 86 suggested by Nick Johnson and Vitalik Buterin [7]. EIP86 permits receivers of a transaction paying the gas fee. This would certainly enable a functional Möbius and Miximus as well, since the tumbling contract could pay for the withdrawal transactions' gas fee, eliminating

the previous workaround to unlinkably fund freshly mixed addresses. Additionally, EIP86 also allows contracts and accounts to define their own digital signature algorithms. This means that users are no longer required to sign transactions with Elliptic Curve Digital Signature Algorithm (ECDSA). Moreover if EIP86 or something similar is implemented, which is expected in 2019, MixEth is also made viable.

F.1 Minimizing shuffle transactions with trusted execution environments

One might effectively minimize the number of necessary shuffling rounds to 1. If a Trusted Execution Environment (TEE), e.g., Intel SGX is used to generate the shuffling transaction, then even a single shuffling transaction would suffice to provide the same level of anonymity as if every participant shuffles the public keys. Any of the participants could upload the TEE-generated shuffling transaction, while the MixEth contract could check that indeed the shuffling transaction was generated by a TEE. Such a shortcut would make our scheme even more practical and efficient, however it would subsume trust in Intel regarding the security and confidentiality of the TEE.