5th International Symposium on Foundations and Applications of Blockchain 2022

FAB 2022, June 3, 2022, Berkeley, CA, USA

Edited by

Sara Tucci-Piergiovanni
Natacha Crooks

Editors

Sara Tucci-Piergiovanni
CEA LIST, Université de Paris-Saclay, France
sara.tucci@cea.fr

Natacha Crooks
University of California, Berkeley, CA, USA
ncrooks@berkeley.edu

ACM Classification 2012
Theory of computation → Distributed algorithms; Computer systems organization → Dependable and fault-tolerant systems and networks; Applied computing → Digital cash; Applied computing → Online banking

ISBN 978-3-95977-248-8

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/978-3-95977-248-8.

Publication date
June, 2022

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.

License
This work is licensed under a Creative Commons Attribution 4.0 International license (CC-BY 4.0): https://creativecommons.org/licenses/by/4.0/legalcode.

In brief, this license authorizes each and everybody to share (to copy, distribute and transmit) the work under the following conditions, without impairing or restricting the authors' moral rights:
- Attribution: The work must be attributed to its authors.

The copyright is retained by the corresponding authors.

Digital Object Identifier: 10.4230/OASIcs.FAB.2022.0

ISBN 978-3-95977-248-8 **ISSN 1868-8969** **https://www.dagstuhl.de/oasics**

OASIcs – OpenAccess Series in Informatics

OASIcs is a series of high-quality conference proceedings across all fields in informatics. OASIcs volumes are published according to the principle of Open Access, i.e., they are available online and free of charge.

Editorial Board

- Daniel Cremers (TU München, Germany)
- Barbara Hammer (Universität Bielefeld, Germany)
- Marc Langheinrich (Università della Svizzera Italiana – Lugano, Switzerland)
- Dorothea Wagner (*Editor-in-Chief*, Karlsruher Institut für Technologie, Germany)

ISSN 1868-8969

https://www.dagstuhl.de/oasics

Contents

Preface
Sara Tucci-Piergiovanni and Natacha Crooks 0:vii

Invited Talks

Reflections on the Past, Present and Future of Blockchain Foundations and Applications
Ittai Abraham 1:1–1:1

Some Insights on Open Problems in Blockchains: Explorative Tracks for Tezos
Sylvain Conchon 2:1–2:1

Hierarchical Consensus: A Horizontal Scaling Framework for Blockchains
Alfonso de la Rocha 3:1–3:1

Efficient DAG-Based Consensus
Alberto Sonnino 4:1–4:1

Regular Papers

Fork Accountability in Tenderbake
Antonella Del Pozzo and Thibault Rieutord 5:1–5:22

Dynamic Blockchain Sharding
Deepal Tennakoon and Vincent Gramoli 6:1–6:17

Posters

Analyzing Soft and Hard Partitions of Global-Scale Blockchain Systems
Kevin Bruhwiler, Fayzah Alshammari, Farzad Habibi, Juncheng Fang, and Faisal Nawab 7:1–7:1

A Modular Approach for the Analysis of Blockchain Consensus Protocol Under Churn
Floris Ciprian Dinu and Silvia Bonomi 8:1–8:2

Improving Blockchain Resilience to Network Partitioning by Sharding
Juncheng Fang, Farzad Habibi, Kevin Bruhwiler, Fayzah Alshammari, and Faisal Nawab 9:1–9:1

Why General Collective Intelligence Must Be the Future of the Blockchain
Andy E. Williams 10:1–10:3

5th International Symposium on Foundations and Applications of Blockchain 2022 (FAB 2022).
Editors: Sara Tucci-Piergiovanni and Natacha Crooks
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Preface

The goal of 5th International Symposium on Foundations and Applications of Blockchain 2022 (FAB'22) is to bring researchers and practitioners of blockchain – the technology behind Bitcoin – together to share and exchange results. The program of FAB'22 features four invited talks, two regular scientific papers, followed by a poster session. The program committee selected two regular papers for publication in the proceedings out of eight submissions.

Prof. Sylvain Conchon's invited talk is about the emergent challenges in blockchains and how they are tackled in the Tezos blockchain. Alberto Sonnino's invited talk is about building high-performance BFT consensus based on DAG protocols. The invited talk of Alfonso de la Rocha is about improving blockchain performances through hierarchical consensus. Finally, Ittai Abraham's talk is about future of blockchains and some exciting opportunities in RegDeFI.

The two regular scientific papers published in these proceedings cover two important emerging topics: sharding and accountability. Deepal Tennakoon and Vincent Gramoli present dynamic blockchain sharding, a new way to create and close shards on-demand, and adjust their size at runtime. Antonella del Pozzo and Thibault Rieutord study approaches to make BFT consensus protocols accountable, considering Tenderbake as a case study.

The proceedings include as well four posters presenting interesting research proposals. Juncheng Fang et al. research proposal is about improving blockchain protocols when recovering from network partitions. Floris Dinu and Silvia Bonomi present a proposal for analyzing and comparing different consensus protocols used in blockchain under churn. Andy Williams presents a proposal to frame blockchain in the scope of collective intelligence. Finally Kevin Bruhwiler et al. propose an approach to study network partitions through simulation.

The program also features a keynote from the Ethereum foundation.

Finally, we thank the authors for providing valuable content, and the program committee who gave very valuable feedback to the authors. We also thank Algorand, Protocol Labs and Ethereum for their financial support.

Sara Tucci-Piergiovanni and Natacha Crooks

5th International Symposium on Foundations and Applications of Blockchain 2022 (FAB 2022).
Editors: Sara Tucci-Piergiovanni and Natacha Crooks
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Reflections on the Past, Present and Future of Blockchain Foundations and Applications

Ittai Abraham ✉
VMware, Herzelia, Israel

Abstract

We survey some of the amazing progress in Blockchain technology in the last 5 years: from foundations like consensus protocols, execution models, and zero-knowledge proofs, to why these foundations are critical for applications like decentralized finance and web3. The main part of the talk will try to envision the future of Blockchains: how will the "Endgame" look like? What foundations are we still missing? We argue that for Blockchains to thrive and reach Billions of users, we should expect a much more regulated landscape to emerge and discuss some exciting opportunities in reg-crypto and RegDeFi. An example of this direction is our new work on UTT which is a decentralized Ecash system with accountable privacy.

2012 ACM Subject Classification Theory of computation → Distributed algorithms; Applied computing → Digital cash; Computer systems organization → Dependable and fault-tolerant systems and networks

Keywords and phrases Blockchain

Digital Object Identifier 10.4230/OASIcs.FAB.2022.1

Category Invited Talk

© Ittai Abraham;
licensed under Creative Commons License CC-BY 4.0

Some Insights on Open Problems in Blockchains: Explorative Tracks for Tezos

Sylvain Conchon ✉
Laboratoire Méthodes Formelles, Université Paris-Saclay, CNRS, ENS Paris-Saclay, 91190 Gif-sur-Yvette, France

Abstract

Blockchain is an emerging field that started with the advent of Bitcoin, the first cryptocurrency launched in 2008. Since then, new distributed applications (DApps) based on blockchain have emerged, such as non-fungible tokens (NFT) or decentralized finance (DeFi). All this contributes to an ever-increasing use of blockchains and poses many technological and scientific challenges.

The first challenge is related to *scalability*, usually measured by the number of transactions per second (TPS) that a blockchain can process. Recent solutions, such as Rollups, implement the concept of Layer 2, a secondary framework built on top of an existing blockchain that allows transactions to be managed off-chain for efficiency. The primary blockchain is used to secure the exchanges of the second layer by regularly recording its exchanges and its current state. A first experiment of Optimistic Rollups has been implemented in the Blockchain Tezos. The TORUs (Transaction Optimistic Rollups) allow efficient financial assets exchanges in the form of Michelson tickets. A generalization to Smart contracts Optimistic Rollups (SCORU) is currently under development.

Another challenge is to improve the *efficiency* of the data structures used in blockchain implementations. The main explorative tracks are to reduce and improve disk usage (compact representations, serialization of big data, sharing, ...), increase the speed of access operations (efficient caching strategies, asynchronous I/O, ...). For example, recent improvements to the storage layer of Octez, Tezos' most popular node implementation, have shown that it is possible to significantly speed up transactions, stabilize average transaction latency, and significantly reduce memory usage.

The *security* issues associated with blockchains also raise many challenges. Indeed, the economic protocols or consensus algorithms implemented in blockchains use incentive mechanisms to discourage nodes from engaging in bad behavior or in launching attacks. A fine tuning of these incentives is difficult in situations where decision makers interact. Game theory can be used to develop incentives, in particular its integration into verification tools (model-checkers, proof assistants, deductive program verification) or machine-learning tools could be very promising.

Finally, given the financial amounts managed by blockchains, it is essential to have a very precise specification of the algorithms, protocols and data structures used in blockchain implementations in order to guarantee the *reliability* of these very complex software. Whether it is for the programming of smart contracts, consensus algorithms or the P2P layer, the introduction of formal methods in the development cycle of blockchains is a major challenge in this domain. A lot of work in formal methods has been done for the Tezos blockchain. Among others, the formalization in TLA+ of Tenderbake, a PBFT-style consensus algorithm which offers deterministic finality to Tezos.

Author Bio. Sylvain Conchon is Professor in Computer Science at University Paris-Saclay since 2013. He is a member of LMF (Formal Methods Laboratory) and his research focuses on automatic deduction and model-checking, using techniques based on SMT (Satisfiabilty Modulo Theories) solvers. He is one of the designers of the SMT solver Alt-Ergo and the model-checker Cubicle. In collaboration with Nomadic-Labs, he is currently working on the use of formal methods to design and verify several aspects related to the blockchain Tezos, such as Michelson smart contracts or the Tenderbake consensus algorithm.

2012 ACM Subject Classification Software and its engineering

Keywords and phrases Blockchain, Tezos, Scalability, Efficiency, Security, Reliability

Digital Object Identifier 10.4230/OASIcs.FAB.2022.2

Category Invited Talk

© Sylvain Conchon;
licensed under Creative Commons License CC-BY 4.0
5th International Symposium on Foundations and Applications of Blockchain 2022 (FAB 2022).
Editors: Sara Tucci-Piergiovanni and Natacha Crooks; Article No. 2; pp. 2:1–2:1
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Hierarchical Consensus: A Horizontal Scaling Framework for Blockchains

Alfonso de la Rocha ✉
Protocol Labs, Madrid, Spain

Abstract

In this talk we present the Filecoin Hierarchical Consensus framework, which aims to overcome the throughput challenges of blockchain consensus by horizontally scaling the network. Unlike traditional sharding designs, based on partitioning the state of the network, our solution centers on the concept of subnets –which are organized hierarchically– and can be spawned on-demand to manage new state. Child subnets are firewalled from parent subnets, have their own specific policies, and run a different consensus algorithm, increasing the network capacity and enabling new applications. Moreover, they benefit from the security of parent subnets by periodically checkpointing state. In this paper, we introduce the overall system architecture, our detailed designs for cross-net transaction handling, and the open questions that we are still exploring.

Author Bio. Before joining Protocol Labs, Alfonso worked as a blockchain expert at Telefónica R&D, where he was responsible for the design and development of core technology based on blockchains, distributed systems, and advanced cryptography. Alfonso's involvement in research and development began at Universidad Politécnica de Madrid, where he worked on topics related to energy efficiency in data centers. His broad R&D experience also includes research into the compression efficiency of video coding standards at Ericsson Research and projects related to securing interdomain routing protocols at KTH Royal Institute of Technology in Stockholm. (Bio link)

2012 ACM Subject Classification Computer systems organization → Distributed architectures; Software and its engineering → Distributed systems organizing principles

Keywords and phrases blockchain, consensus, distributed systems, P2P, scalability, sharding

Digital Object Identifier 10.4230/OASIcs.FAB.2022.3

Category Invited Talk

Related Version *Full Version*: `https://research.protocol.ai/publications/hierarchical-consensus-a-horizontal-scaling-framework-for-blockchains/delarocha2022.pdf`

© Alfonso de la Rocha;
licensed under Creative Commons License CC-BY 4.0
5th International Symposium on Foundations and Applications of Blockchain 2022 (FAB 2022).
Editors: Sara Tucci-Piergiovanni and Natacha Crooks; Article No. 3; pp. 3:1–3:1
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Efficient DAG-Based Consensus

Alberto Sonnino
Mysten Labs, London, UK

Abstract

This talk shows how to build high-performant Byzantine fault-tolerant (BFT) quorum-based consensus cores. The talks starts by challenging the common misconception that the overall communication complexity of the protocol is the key factor determining performance. We instead argue that the bottleneck of many state-of-the-art consensus protocols is their sequential use of the machine's resources (network, storage, CPU), and that data dissemination is the most resource-intensive task.

In light of the above considerations, the first insight to build performant BFT-based consensus cores is to separate the task of reliable transaction dissemination from transaction ordering. We show how to design a new DAG-based mempool protocol, called Narwhal, specialising in high-throughput reliable dissemination and storage of causal histories of transactions. Narwhal tolerates an asynchronous network and maintains high performance despite failures. It is designed to easily scale-out using multiple workers at each validator to concurrently use the machine's resources (network, storage, CPU), and demonstrates that there is no foreseeable limit to the throughput we can achieve. We then present two ways to leverage Narwhal to achieve consensus. We first (i) present Tusk, a zero-message overhead asynchronous consensus protocol designed to work with Narwhal. Tusk achieves an unprecedented 160,000 tx/s with about 3 seconds latency in a geo-replicated environment. We then (ii) show how any partially-synchronous consensus, such as HotStuff (PODC 19), can be composed with Narwhal to drastically improve its performance. HotStuff running over Narwhal sees its throughput increase from about 2,000 tx/s to over 130,000 tx/s without noticeable latency increase.

The talk concludes by illustrating how to properly evaluate performance of BFT-based consensus cores. It highlights the most common mistakes seen in the literature, such as benchmarks with empty transactions (empty load), performance approximation based on LAN-only benchmarks, and using a single burst of input transactions. We then show how to analyse benchmark results using latency-throughput graphs (L-graphs) and SLA-based throughput graphs.

Author Bio. I am a system researcher at Mysten Labs, based in London (UK). I previously was a research scientist at Facebook (now called Meta) in the blockchain and cryptography team. Before joining Facebook, I co-founded chainspace.io which built a scalable smart contract platform; the team was then acquired by Facebook. My research interests are in systems security and privacy engineering. My main areas of research include distributed systems, blockchains, and privacy enhancing technologies. I have a special interest in cryptography, and I spend most of my time designing, implementing and evaluating high-performance distributed systems.

2012 ACM Subject Classification Security and privacy → Distributed systems security

Keywords and phrases Consensus protocol, Byzantine Fault Tolerant

Digital Object Identifier 10.4230/OASIcs.FAB.2022.4

Category Invited Talk

Funding The majority of this work has been done when the authors were part of the Novi team at Facebook.

Acknowledgements This talk is based on the paper "Narwhal and Tusk: A DAG-based Mempool and Efficient BFT Consensus" (EuroSys 22) authored by George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman.

© Alberto Sonnino;
licensed under Creative Commons License CC-BY 4.0
5th International Symposium on Foundations and Applications of Blockchain 2022 (FAB 2022).
Editors: Sara Tucci-Piergiovanni and Natacha Crooks; Article No. 4; pp. 4:1–4:1
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Fork Accountability in Tenderbake

Antonella Del Pozzo ✉
Université Paris-Saclay, CEA, List, F-91120, Palaiseau, France

Thibault Rieutord ✉
Université Paris-Saclay, CEA, List, F-91120, Palaiseau, France

Abstract

This work investigates the Fork Accountability problem in the BFT-Consensus-based Blockchain context. When there are more attackers than the tolerated ones, BFT-Consensus may fail in delivering safety. When this occurs, Fork Accountability aims to account for the responsible processes for that safety violation.

As a case study, we consider Tenderbake when the assumption on the maximum number of Byzantine validators – participants involved in creating the next block – does not hold anymore. When a fork occurs, there are more than one-third of Byzantine validators, and we aim to account for the responsible validators to remove them from the system. In this work, we compare three different approaches to implementing accountability in the case of a fork. In particular, we show that in the case of a fork, if we do not modify Tenderbake or we enrich it with a reliable broadcast communication abstraction, then we can account Byzantine processes only in particular scenarios. Contrarily, if we change Tenderbake such that the exchanged messages also carry extra information (which size is proportional to the duration of the current consensus computation), then we can account for Byzantine processes in all kinds of scenarios; however, at the cost of unbounded message size and unbounded local memory.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Blockchain, BFT-Consensus, Fork Accountability

Digital Object Identifier 10.4230/OASIcs.FAB.2022.5

Funding This work was founded by Nomadic Labs.

Acknowledgements The authors warmly thank Lăcrămioara Aştefănoaei, Eugen Zălinescu and Sara Tucci-Pieriogvanni for all the insightful discussions that improved the quality of this work.

1 Introduction

A Blockchain, as the name suggests, is a chain of blocks. Current Blockchain solutions are divided into blockchains with probabilistic finality and blockchains with immediate finality. The most known blockchains, Bitcoin [18] and Ethereum v1.0 [20], are based on the Proof-of-Work mechanism to decide on the next block to append, and in that case, they provide probabilistic finality. Once a block appears in the $i-th$ position of the blockchain, it will stay there with a probability that exponentially grows proportionally to the length of the chain extending it [14]. In the case of immediate finality, as in the case of Tendermint [6] and Tenderbake [1], we have that a new BFT Consensus instance is run to decide on the next block to append. Hence, once a block appears in the $i-th$ position it stays there forever. However, BFT Consensus works as long as, given a set of n committee members (or validators) in charge to decide for the next block, at most $f = n/3 - 1$ are affected by Byzantine failures (validators showing arbitrary behaviors). As long as this assumption holds, we guarantee that precisely one block is decided for each consensus instance. Contrarily, if the assumption is violated, the blockchain can experience forks (loss of safety) or interruption of block production (loss of liveness).

© Antonella Del Pozzo and Thibault Rieutord;
licensed under Creative Commons License CC-BY 4.0
5th International Symposium on Foundations and Applications of Blockchain 2022 (FAB 2022).
Editors: Sara Tucci-Piergiovanni and Natacha Crooks; Article No. 5; pp. 5:1–5:22
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Nevertheless, it is ambitious to assume that at most f validators are affected by Byzantine failures in a long-running system. Especially if we think that in the current solutions, we potentially select a new committee for each new block, and there could be mistakes in this selection process from time to time. On the contrary, if we do not frequently change the committee members, then bribery attempts might occur. To this end, since most of the current selection mechanisms are based on the stake held and bonded by validators, the bonded stake of misbehaving validators can be slashed in case of provable misbehavior. It plays as a disincentive to misbehave. In such a context, producing proof of misbehavior is a key aspect in putting in place actions to disincentivize validators from misbehaving. Once we have them, such proofs can be used to perform slashing actions. The procedure of deriving non-reputable proofs of misbehavior is known as accountability [16].

In this work, we consider accountability when more than f processes in a committee are Byzantine faulty and produce a fork. In this case, we refer to it as the fork accountability problem. Similarly, as in [12], in case of loss of safety, we aim to identify at least $f+1$ of the byzantine processes responsible for the fork that occurred. We consider as a case of study Tenderbake [1], a BFT-Consensus protocol for the Tezos Blockchain [15, 3] designed for an eventually synchronous system in which processes are equipped with bounded buffer and can experience messages loss before the beginning of the period of synchrony.

Our contribution is the following. First, we organize the different scenarios that can induce a fork in Tenderbake in a fork taxonomy. Secondly, we design and compare three accountability approaches: the first considers $Tenderbake$ as it is and investigates in which scenarios we can achieve accountability; in the second approach, we show how to modify Tenderbake to enrich messages with the necessary information to achieve accountability but at the price of unbounded message size; in the third approach, consider an approach in the middle of the previous two. We add reliable communications (contrarily to the previous two approaches) and some extra information in the messages. Interestingly, we obtain the same results as in the first approach. In the rest of the paper, we replace the f standard notation with T to design the maximum number of tolerated Byzantine processes. We use f_i to design the actual number of Byzantine processes during the consensus instance i, such that $f_i < n$, where n is the number of processes in the committee.

Related Work. Distributed system research areas are more focused on failure detectors [10, 5] than accountability. While the failure detector aims to provide each system process with the identities of faulty processes to help them make progress in the computation, accountability aims to provide non-repudiable proofs that can be shared with other processes. It plays a crucial role in putting in place countermeasures against faulty processes in a distributed system. As pointed out in [2], if processes are accountable for their behavior, then rational processes have an incentive to behave correctly.

To the best of our knowledge, PeerReview [16] is one of the first works that proposed a general solution to have accountability in distributed systems. They developed a system called PeerReview that implements accountability as an add-on feature for any distributed system. Each process in the system records messages in tamper-evident logs; an auditor can challenge a process, retrieve its logs, and simulate the original protocol to ensure that the process correctly behaved. They show that in doing so, it is always possible to identify at least one Byzantine process (if some process acts in a detectable Byzantine way). The main issue is that an auditor, to prove that a process is Byzantine, must receive a response from it. If no response is received, then the auditor cannot determine whether the process is Byzantine or whether the network has not yet stabilized. It follows that the Byzantine will

only be suspected forever but not proved guilty. This limitation is common to distributed protocols that are not designed to provide accountability. That is also true in the blockchain context, we observe that protocols as Tenderbake [1] and Tendermint [17] suffers from the same problem. Indeed, even though it is possible to observe deviation from the protocols, it might be necessary to prove the absence of messages to prove a party is guilty of bad behavior. For example, in a protocol, a message m is triggered on the reception of message m' from a quorum of processes. If m is triggered without the reception of sufficiently many occurrences of m', then the only way to prove that m is an expression of bad behavior is to prove that the causally dependent messages m' do not exist.

Polygraph [12] is the first work that manages to have accountability in the case of Byzantine behaviors, circumventing this issue by using message justifications. Contrarily to PeerReview, they do not use an accountability module as an add-on feature for a distributed protocol; they designed a distributed protocol that provides accountability features by design. Intuitively, messages sent during the computation carry the necessary information to have accountability in the case of forks. In case of failures, there is no need to query processes to obtain proof of their innocence.

Unlike adding extra accountability information in the protocol's messages, as Polygraph does, another accountability approach is to add extra information directly into the blockchain itself. While not done for accountability purposes, Streamlet [9] creates blocks in each round of computation while providing finality only when particular conditions are met. In other words, in Streamlet processes add blocks to a blocktree, and the fact that a block is added to the tree does not mean that such block is final. This allows the blocktree to be formed of non-finalized concurrent branches keeping track of the evolution of the computation. For finalization, Streamlet applies a rule to the tree to select the finalized branch. As a side result, it has been showed that Streamlet provides accountability without the need to add extra information, either on chain or in the exchanged messages [4]. Indeed, if two concurrent branches manage to be finalized, then the faulty behavior can be traced back to conflicting blocks up to the responsible processes. The work in [4] provides a specification of Accountability for Streamlet. When we compare the information that processes keep locally, we observe that by default, nodes keep intermediary unconfirmed blocks contrarily to Tenderbake. This provides an interesting trade-off between the amount of information and the Accountability capabilities.

In the Proof-of-Stake blockchain context, accountability represents an important goal. The idea is that users that violate the protocol can be punished by confiscating their deposited stake (e.g., Casper [7]). Tendermint as well is going in that direction [1]. However, there is not yet a solution specification that exhaustively performs fork accountability, encompassing all scenarios. The work of Sheng et al. [19] aims at modifying existing blockchain BFT consensus protocols to enforce them with fork accountability. In particular they consider PBFT [8], HotStuff [21] and Algorand [11]. Interestingly, a very recent work, [13] presents how to turn any consensus protocol into an accountable one with two additional all-to-all communication steps. It is important to stress that in the case of Tenderbake, we have to face an extra difficulty given by the lossy nature of channels, which poses an extra challenge in collecting proofs in case of misbehavior compared to the existing works.

[1] The fork accountability was described in the Cosmos blog `https://v1.cosmos.network/resources/whitepaper#fork-accountability` as in Tendermint discussions `https://github.com/tendermint/tendermint/issues/4189`. In the Tendermint documentation, we can find an analysis of the kind of Byzantine misbehavior that we can observe with respect to the protocol execution `https://docs.tendermint.com/master/spec/light-client/accountability/#on-chain-attacks`.

Our contribution. In this document, we adapt the misbehavior analysis in the Tendermint context to Tenderbake [1], and we derive a fork taxonomy. Furthermore, we establish that with the current version of Tenderbake, we can only account for Byzantine processes for a subset of the possible fork typologies. Nevertheless, if we consider an extended version of Tenderbake enriched with message justifications, we can derive accountability conclusions with any kind of forks. The price to pay is the requirement of unbounded local memory due to the unbounded size of messages. This result makes a step forward in understanding the cost of Fork Accountability depending on the kind of protocol.

Paper organization. The work is organized as follows, in Section 2 we define the system model as an extension of the system model for Tenderbake [1]. Section 3 recalls how Tenderbake works, and Section 4 organizes the different kinds of forks in a fork taxonomy and discusses the difficulty of having accountability in different cases. In Section 5.1 we define the Fork Accountability problem and we discuss how to solve it in the following cases: if we can collect all exchanged messages before a fork (Section 5.2); if we consider Tenderbake as it is (Section 3); if we enrich Tenderbake with message justifications (Section 5.4) and finally; if, contrarily to Tenderbake in [1], we have reliable communications (Section 5.5). In Section 5.6 we compare and discuss the trade-offs between the accountability and the message costs of the presented approaches, and finally, in Section 6 we discuss future directions.

2 System model and definitions

In this work, we refer to Tenderbake as defined in [1]. Briefly, Tenderbake solves Dynamic Repeated Consensus, i.e., it executes consensus instances in sequence to produce an infinite output of decided values. Each consensus instance is executed by a (possibly) different committee of n processes. We assume that in the committee of processes running a consensus instance, T is the maximum number of Byzantine processes, and the number of committee members is at least $n = 3T + 1$. For each committee c_i, we refer to f_i as the number of Byzantine faulty processes present in such a committee. Additionally, for each committee c_i, if $f_i \leq T$ then we say that c_i is a *correct committee*, otherwise c_i is a *Byzantine-faulty committee*. Processes have access to digital signing and hashing algorithms. For simplicity, we assume that cryptography is perfect: digital signatures cannot be forged, and there are no hash collisions. Each process has an associated public/private key pair for signing, and processes can be identified by their public keys.

We consider the same assumptions as defined in the system model of Tenderbake [1]. In particular, we assume a partially synchronous system, in which after some unknown time τ (the global stabilization time, GST) the system becomes synchronous and channels reliable. That is, there is a finite *unknown* bound δ on the message transfer delay. Before τ, the system is asynchronous, and channels are lossy. We assume the presence of a best-effort broadcast primitive used by processes participating in a consensus instance. Broadcasting messages is done by invoking the primitive broadcast. This primitive provides the following guarantees: (i) integrity, meaning that each message is delivered at most once and only if some process previously broadcasts it; (ii) validity, meaning that after τ if a correct process broadcasts a message m at time t, then every correct process receives m by time $t + \delta$. This communication primitive is built on top of point-to-point channels, where exchanged messages are authenticated. When specified, we consider a reliable broadcast abstraction built on top of no lossy channels, which provides integrity, validity, and agreement. The

integrity property does not change, and contrarily to the best-effort broadcast, the validity became, if a correct process broadcasts a message m at time $t > \tau$, then it receives it. The agreement property guarantees that if a correct process receives m at time $t > \tau$, every correct process eventually receives it by time $t + \delta$.

In the following, we consider that processes are executing a protocol for solving Dynamic Repeated Consensus (DRC) as defined in [1]. Informally, given that processes have an infinite sequence of input values, DRC guarantees the following three properties: (i) progress: each correct process has an infinite sequence of output values; (ii) validity: for each correct process, the sequence of output values satisfies a predetermined predicate isValid(); and (iii) agreement: at any time, for any two sequences of outputs at correct processes, one is the prefix of the other. In the rest of the document, we use the terms agreement and safety interchangeably, and the same with the terms progress and termination.

3 Tenderbake and Round-based BFT-Consensus protocols

In this section, we present the main structure of Tenderbake. We start with a general overview of a round-based BFT-Consensus protocol, and we give more details on the specifics of Tenderbake, which is at the base of DRC. For a complete description of Tenderbake we refer to [1]. Round-based BFT-Consensus protocols are executed by a set of n processes, such that one process per round is selected as proposer. The proposer is in charge to drive all the other correct processes to agree on the same value at the end of that round. If this is not the case, a new round begins with a new proposer. To avoid safety violation between one round and another, processes carry certain information from one round to another. Given communication delay, we could have that only some correct process decides for a value in a round r, which implies that all the other correct processes have to agree on that value in the subsequent rounds. In most cases such as Tenderbake and Tendermint, safety is preserved thanks to the locking mechanism.

In particular, Tenderbake works as follows. Each round is divided into three sequential phases: PROPOSE, PREENDORSE, and ENDORSE. During the PROPOSE phase, only the unique designated proposer proposes to the committee members a single value b (a block proposal), either a new value or one inherited from a previous round. During the PREENDORSE phase, a process preendorses b if b comes from the designated proposer and if the process is not already locked or if it is locked on an outdated value. During the ENDORSE phase, if processes receive a preendorsement from a quorum for b, they lock on it and endorse it. Finally, if processes receive an endorsement from a quorum for b, they decide b. Let us stress the role of the lock variable. Such variable is set to a value b when potentially there could be some processes about to decide on it, and it is set to another value $b' \neq b$ only when a process observes evidence that no correct process might have decided on b. Each process signs the messages it sends, and each message carries a value b associated with the round and the phase. In the same spirit, each proposed block b is labelled with the round and the proposer that proposes it and became decided when there exists a Quorum Certificate (QC) of messages labelled with the same round r and the phase ENDORSE from $2T + 1$ different processes that refers to b. Each block b is composed of a block header and a block payload. In this context, we are not interested in the content of a block payload. Each block has the pointer to its (unique) predecessor block in the block header. If there are two different decided blocks with the same predecessor, we have a fork (safety violation).

4 Fork Taxonomy

This section aims at analyzing how forks can occur when committees are byzantine. The result is a fork taxonomy whose purpose is twofold. First, it helps in designing an accountability module: to collect evidence of Byzantine activities and make them accountable for their actions (if possible). Second, it helps in understanding the impact of Byzantine committees. Nevertheless, it is also the first step to designing fork recovery strategies, which is out of the scope of this document.

4.1 Fork Taxonomy

In the following, we define the kind of forks that can occur when Tenderbake runs under the hypothesis of more than T Byzantine failures in some committee c_i, in particular, $T < f_i$. Notice that the same can be applied to other repeated consensus protocols based on the locking mechanism and rotating coordinator (proposer) paradigm. Let us briefly recall that each consensus instance proceeds in rounds. Each round has a different proposer, and specific information is carried by processes from one round to another, such as the locking variable, to prevent the safety violation. In such context, we distinguish two kinds of forks, Intra-round forks, when two or more valid blocks[2] are produced during the same round, and Cross-round forks, when two or more valid blocks are produced across different rounds due to the violation of the locking mechanism. Finally, we present the Cross-committee fork, which occurs when we allow multiple committee selections for the same height (for instance, to deal with the absence of valid block production). Let us remark that this fork cannot occur with the current version of Tenderbake, and we discuss it for completeness. Given a Byzantine committee c_i, we define the following kinds of forks that can occur. In particular, if we have $T < f_i \leq 2T$ then only safety can be violated (or liveness, but there are no forks), while if $2T < f_i \leq n$, then both safety and validity can be violated.

- **Intra-Round (IR) fork ($T < f_i \leq 2T$)**: the fork is produced during the same round, i.e., there are at least two valid blocks under the same proposer. All the blocks in this fork share the same committee, proposer, round, and the same height;
- **Full Byzantine Intra-Round (FBIR) fork ($2T < f_i \leq n$)**: the fork is produced during the same round without any needed participation from correct committee members. Notice that we could also have a non-compliant block with the application level in this case even though it is valid in the sense that it carries a CQ, e.g., a block proposed by a Byzantine that is not the current proposer for that round. (Validity property violation)
- **Cross-Round (CR) fork ($T < f_i \leq 2T$)**: the fork is produced during different rounds, i.e., there are at least two valid blocks proposed during two distinct rounds and potentially distinct proposers. All the blocks in this fork share the same committee and height.
- **Full Byzantine Cross-Round (FBCR) fork ($2T < f_i \leq n$)**: the fork is produced during different rounds without any needed action from correct committee members. Notice that in this case, as before, we can have the Validity property violation.

[2] A block is said to be valid if it comes with a Quorum Certificate (CQ) of $2T + 1$ different endorsement (votes, or precommit – depending on the protocol vocabulary) messages from the same round and height.

4.1.1 Discussion on accountability

Let us recall that if there is a fork induced under the same committee, then we can have two scenarios, IR and CR forks. Interestingly, let us consider the first version of Tenderbake [1], in the case of (FB) IR forks. We can produce accountability proofs for any IR forks, considering the blocks' information. However, in the case of CR, the information in the blocks is not enough. More details are below.

- IR-Fork. In this case, the proposer for the round is Byzantine and proposes more than one block such that correct processes endorse only the first one they are aware of, and Byzantines endorse both. Any block in the fork comes with a QC of $2T+1$ endorsements. It follows that any pair of blocks share at least $T+1$ endorsements and up to $2T$. Those are the accountability proof for at least $T+1$ Byzantine committee members. The proposer is also trivially accountable, and any pair of blocks give the proof of the fork.
  - FBIR-Fork, in this case, we can apply the same reasoning as for the IR-Fork but contrarily to it, two blocks can share the totality of endorsements, up to $2T+1$. In this case, one of the proposers might be correct (the round proposer if it proposed only one of the two blocks).
- CR-Fork. The fork is composed of blocks produced during different rounds. In this case, the fork is due to the locking mechanism violation, i.e., there is some correct process that locks for different valid blocks (instead of at most one), not being aware that a previously preendorsed and endorsed block was decided (collecting $2T+1$ endorsements). In this case, accountability is not possible by solely using the block's information. Indeed, we have that pairs of valid blocks can share the validators that signed the endorsements in their QC. For such a reason, Tenderbake has to be modified to gather enough information to distinguish between correct committee members that endorsed multiple times from Byzantine committee members. We discuss those modifications in the next sections.
  - FBCR-Fork, in this case, the lock does not have to be violated. Byzantine processes can directly produce two valid blocks by endorsing a proposal (from a valid proposer or not). This case inherits the same limitation as the previous one.

5 Fork Accountability

In this section, we define the Fork Accountability problem. Hereafter, we provide a pedagogical solution with all the available information (i.e., messages). Later we move to the specific case of the restricted information available with Tenderbake to discuss the limitation of the accountability accuracy and completeness that we can get. We further design modifications to Tenderbake to satisfy Fork Accountability. Finally, we investigate how improved communication primitives impact Fork Accountability.

5.1 Fork Accountability problem definition

An Algorithm A (Tenderbake in our case) is modeled as a collection of n deterministic automata, where $A(i)$ specifies the behavior of process i. Computation proceeds in steps of this automata. In each step (i, m, A) a process i first receives a message m or accepts an input (internal or external event) and after it changes its states according to $A(i)$. Finally, i sends a message specified by $A(i)$ for the new state to processes or produces an event. Let $E(A)_i$ be an execution of A at process i as the sequence of steps executed by i. Finally, let $M(E(A)_i)$, for conciseness M_i, be the set of messages m received by process i during an execution of A.

We define the accountability module for Tenderbake in terms of completeness and accuracy properties. Such module takes as input the messages M_i received by a correct process i during the execution of Tenderbake. If a fork occurs, it outputs the faulty participants and proof of their responsibility in producing such a fork.

- (**Completeness**) if a fork occurs then at least $T+1$ committee members are accountable as faulty;
- (**Accuracy**) no correct process is ever accountable as faulty.

The Completeness property is similar to the Accountability property of the Accountable Byzantine Agreement problem as defined in [12] which merges the Fork Accountability and the BFT-Consensus problems.

5.2 Fork Accountability with all messages

For pedagogical purposes, in the following, we describe how to perform accountability if a correct process has access to the whole set of messages exchanged during an execution i of Tenderbake, M_i. More in detail, given the occurrence of a fork, a process can collect all the messages exchanged between correct processes before that fork occurs, as if they have been transmitted after τ by reliable and timely communication abstractions. Let us remember that the protocol proceeds in rounds and each round is composed of three steps. During each step, each correct committee member sends at most one message.

In the case of an IR fork, the proposer is necessarily Byzantine. Indeed, for each round, processes consider only the proposer's proposed values, which is supposed to propose a single value. Hence if there is an IR fork, there are not sufficiently many Byzantine processes to issue a fork by themselves, then the proposer proposed at least two different values. Moreover, the committee is Byzantine as well because there must have been sufficiently many Byzantine committee members who endorsed twice on different block proposals. In that case, given a fork at round r, the Byzantine members are detected by selecting from M_i all the members that sent more than one message labeled with round r and phase PROPOSE (more than one block was proposed), and that sent more than one message labeled with round r and phase ENDORSE (Byzantine validators endorsed twice and for different blocks). In the case of an FBIR fork, it can happen that the selected proposer did not propose a block in the fork for that round. In such a case, it means that all processes in the QC are byzantine.

Before digging into the CR fork, let us first describe how it may occur, detailing the faulty flip-flopping scenario [3] which provokes a violation of the locking mechanism and may result in a fork. Interestingly, with such a fork, the information carried in the QC of the decided blocks is not enough to perform fork accountability. Let us first make some observations about Tenderbake:

- A correct committee member decides for a block b associated to a round r if it receives a QC, $2T+1$ endorsements, for it.
- Each correct committee member endorses at most once during a round r, while byzantine-faulty committee member can endorse for an unbounded number of different blocks.
- When a correct committee member i endorses for block b at round r, it also locks for block b at round r.

[3] https://docs.tendermint.com/master/spec/light-client/accountability/#flip-flopping

- The same correct committee member i re-locks (and endorses) for another block b' if i receives a proposal for b' at round r_2 and i already received $2T+1$ preendorsements for b' produced at round r_1, $r < r_1 < r_2$ (cf. Tenderbake [1]).

This means that, if i decided for b but the committee is Byzantine then it can exist some round $r_1 > r$ where there are $2T+1-f_i$ correct committee members that preendorse b' along with f_i Byzantine committee members, then, at round r_2, i can re-lock on b' and decide for it. Hence, the locking mechanism is violated. In that scenario, blocks b and b' can share endorsements signed by the same committee members, either correct or Byzantine. In that case, looking only at the QC of b and b' it is not possible to distinguish correct from Byzantine processes and to perform any fork accountability.

Let us now generalize the flip-flop scenario. Let us consider an execution $E(A)$ of Tenderbake and let B the sequence of all blocks that obtains a QC during $E(A)$. Let S_B be the set of $T+1$ Byzantine committee members and let S_L be a set of T correct committee members that lock and re-lock on all the blocks in B, and finally, let S_U be the set of correct committee members that never lock.

- At round r_k block b is proposed. S_B and S_L pre-endorse b. Processes in S_L, contrarily to processes in S_U, receive the $2T+1$ pre-endorsements for b and then endorse and lock on it. Processes in S_B can assemble the QC for the decided block b with the $2T+1$ endorsements (T from processes in S_L and $T+1$ from processes in S_B) and delay its diffusion for as long as they wish.
- At round r_{k+1} block b_1 is proposed. S_B and S_U pre-endorse it and processes in S_L do not, because already locked. Processes in S_L receive all the $2T+1$ endorsements for b_1.
- At round r_{k+2} block b_1 is proposed again. This time, processes in S_L can pre-endorse b_1 along with S_B (because they get the $2T+1$ pre-endorsements for b_1 during the previous round). Processes in S_L receive the $2T+1$ pre-endorsements for b_1 and then endorse and lock it. Processes in S_B can assemble a valid block b_1 with the $2T+1$ votes (T from processes in S_L and $T+1$ from processes in S_B) and delay its diffusion for as long as they wish.

In this case, in M_i there might not be any committee member that sent more than one message during the same step. A correct committee member does flip-flopping from b to b_1 when it receives enough pre-endorsements for b_1 from a previous round (still greater than the round in which it locked on b). On the contrary, a Byzantine committee member can flip-flop without needing those messages that justify its action. Thus, in order to account Byzantine committee members processes must look for unjustified flip-flopping, i.e., a committee member that endorsed a block b at round r_k and preendorsed b_1 at round r_{k+2} without the existence of $2T+1$ preendorsements for b_1 at a round r_{k+1}, with $r_k < r_{k+1} < r_{k+2}$. In the described scenario, the withheld QC for b at round r_k (that contains the endorsement messages labeled with r_k issued by processes in S_B) plus the preendorsement for block b_1 at round r_{k+1} issued by processes in S_U constitute a proof of processes in S_U performing a faulty flip-flop (as we discuss, a legal flip-flop needs at least three rounds). The origin of the faulty flip-flop stands in the lock violation of Byzantine processes. However, it might not always be so direct to detect it. In the next Sections, we will discuss how to generalize this approach such that it is always possible to find a proof. This analysis would be possible combining the QCs of the decided blocks and M_i if M_i contains all messages ever exchanged during the protocol execution. Unfortunately, this is not possible with Tenderbake given the message lossy nature of the communication channels before τ, the reason why after that, we discuss how information can be added to the QC to perform CR Fork Accountability.

5.3 Partial Fork Accountability with Tenderbake

In the case of Byzantine committee, Tenderbake can incur (FB)IR and (FB)CR forks. In the first case, as we already discussed, accountability is straightforward. The information that we are interested in a block b_i is the proposer and the quorum certificate (CQ) signatures that come with the round in which they were issued. We use the following notation: $\mathsf{sign}(block) = \langle proposer, \{member_1, \ldots, member_{2T+1}\}, round\rangle$. Concerning forks, we use the following terminology: we say that two[4] blocks b_i, and b_j is a fork if they are two blocks of the blockchain with the same parent.

In case of an invalid block in an FBIR or FBCR fork, that is, with a round that does not correspond with the block proposer, then the proposer and all committee members that endorsed the block are byzantine processes, that is, $\mathsf{sign}(b_j)$.members and $\mathsf{sign}(b_j)$.proposer.

In the case of an IR fork, each correct committee member sends only one message per phase in each round. It follows that, for each two pairs of blocks b_i, b_j in a fork, the faulty processes results from the intersection of $\mathsf{sign}(b_i)$.members and $\mathsf{sign}(b_j)$.members. Thus the proposer and at least f committee members are accountable for the fork.

In the case of a CR fork there are no blocks in the same fork sharing the same round and the intersection of $\mathsf{sign}(b_i)$.members and $\mathsf{sign}(b_j)$.members is different than $\emptyset$. However, this gives us no clue about the faulty processes. Let us consider the scenario described in the previous section concerning the faulty flip-flopping. This scenario originates in blocks having QC sharing the same $2T + 1$ signatures. We cannot distinguish among them which signatures belong to correct committee members (if any) and which to Byzantine ones.

To discern correct from Byzantine, we need to combine the information carried by blocks with the exchanged messages. Indeed, a correct committee member does flip-flop from b_i to b_j when it receives enough pre-endorsements for b_j from a previous round (still greater than the round in which it locked on b_i). In contrast, a Byzantine committee member flip-flops without having those messages that justify its action. Moreover, let us recall that no pre-endorsement messages are recorded in the block. Thus those messages need to be kept locally to perform accountability in case of a fork. Therefore, ideally, all messages exchanged during the computation need to be saved. However, committee members do not rely on reliable communication channels, and not all messages are diffused reliably before τ.

In the following, we describe different approaches to provide the accountability module with the necessary information.

5.4 Full Fork Accountability with piggyback Tenderbake

The idea of the solution is relatively simple. Processes produce justifications when locked on a value b and pre-endorse or endorse a different value $b' \neq b$; for shortness in the following, we say that processes do not follow their lock. Notice that being locked on a value b is a local event. However, in the case of a fork involving b we might observe an endorsement for b in a QC. Indeed, a faulty flip-flopping occurs after some processes ignored their lock and preendorsed another value incorrectly. The issue is determining which processes ignored their lock incorrectly from those who followed the protocol. To distinguish between the two, we propose to provide justifications for these preendorsements votes.

When a process is locked but preendorses another value that was later proposed, if correct, it furnishes a quorum of preendorsement messages for the given value. It forms a justification when a process is not following a lock value. All these justification sets are kept in memory

[4] We consider for simplicity the case in which two blocks compose forks, but it can be easily generalized.

and are added to preendorsement and endorsement messages. That is if a process p is locked at round i and does not follow its lock at round $i+k$, then the justifications set of the preendorsement (endorsement) message contains the first preendorsement quorum certificate for a round between $i+1$ and $i+k-1$. Given a fork composed of two blocks, the justification information and the two blocks' endorsement messages should make it possible to account for at least $f+1$ byzantine processes. However, this information alone is not enough when the processes that signed the second block were not directly involved in the flip-flopping. Thus, the justifications must also contain transitive justifications. More into details, if p is induced to flip-flop thanks to the messages from q, then p re-transmits also the justifications for the message from q. Appendix A provides a detailed explanation of the intuition behind our solution. In particular, Appendix A.1 describes a first approach of piggybacking, and Appendix A.2 details which case the previous approach does not work and introduces the transitive justifications.

5.4.1 Description of the modification to Tenderbake

Figures 1–2 detail the Tenderbake specification as described in [1] plus the modifications for the piggybacking (in red in the pseudo-code). It is out of the scope of this work to explain the Tenderbake functioning. In the following, we provide details useful for the fork accountability. Nevertheless, an interested reader can find more details in [1]. Processes have an extra variable $justification_p$, a list of triples of the following type $\{LR : int, LV : prop, peQC : set\ of\ messages\}$. The purpose is to keep for each Locked Value LV and Locked Round LR at process p the justification $peCQ$ that allows the flip-flopping. $peCQ$ is compounded of $2T+1$ pre-endorsement messages for the new locked value received after round LR and before the new locked round. Moreover, pre-endorsement and endorsement messages have an extra field $justify_p$ that is composed of a list of $peQC$ extracted from the third element of the triples in the list $justification_p$. Figure 3 describes the Fork Accountability module that given two decided blocks, returns the set of faulty processes accountable for that. The proof of their accountability is given by the blocks themselves. As auxiliary functions we define $\mathsf{round}(peQC)$ and $\mathsf{value}(peQC)$ that return respectively round and the value associated to the $peQC$ provided as input.

This solution pays the cost of coping with a lossy channel. This cost is in terms of space complexity as message size becomes unbounded with the justifications. This set increases each time a lock occurs during the execution, which can, unfortunately, happen an infinite number of times (it depends on the Byzantine strategy, they can make processes flip-flop infinitely many times).

For detection, we have that when two distinct blocks are produced, processes can compare the endorsements quorums to detect at least $f+1$ byzantine processes. It is done as follows: Let b_1 be the block with the smallest associated round, and let b_2 be the one associated with the greatest round. The endorsement of b_1, i.e., $\mathsf{sign}(b_1).\mathsf{members}$, are then compared with a refinement of the endorsement of b_2, i.e., $\mathsf{sign}(b_2).\mathsf{members}$. The refinement consists of taking endorsements and replacing them with the justification corresponding to the smallest round greater than $\mathsf{sign}(b_1).\mathsf{round}$, if any.

In the particular case of an FBCR fork, we can have the case in which no correct committee members are involved in the fork and thus in any information carried by each block. In this case, indeed, each endorsement comes with a justification produced by other Byzantine committee members. The detection is then performed in the same way as for CR fork, looking at the first justification in b_2 inconsistent with the CQ information of b_1.

```
1   var justification_p = empty list

2   proc handleConsensusMessage(msg)
3     let type_q(ℓ, r, h, payload) = msg
4     if ℓ = ℓ_p ∧ h = h_p ∧ (r = r_p ∨ r = r_p + 1) then
5       if isValidMessage(msg)
6         messages_p := messages_p ∪ {msg}
7         updateEndorsable(msg)
8     else if ℓ > ℓ_p then
9       pullChain

10  proc updateEndorsable(msg)
11    if |preendorsements()| ≥ 2f + 1 then
12      endorsableValue_p := proposedValue()
13      endorsableRound_p := r_p
14      preendorsementQC_p := preendorsements()
15    else if type(msg) ≠ Preendorse then
16      (eR, eV, pQC) := endorsableVars(msg)
17      if eR > endorsableRound_p then
18        endorsableValue_p := eV
19        endorsableRound_p := eR
20        preendorsementQC_p := pQC

21  proc endorsableVars(msg)
22    let pQC = match msg with
23    | Propose_p(ℓ_p, r_p, h_p, (eQC, hu, eR, pQC)) → pQC
24    | Preendorsements(ℓ_p, r_p, h_p, pQC) → pQC
25    return (roundQC(pQC), valueQC(pQC), pQC)

26  proc filterMessages()
27    messages_p := messages_p \ {type(ℓ, r, h, payload) ∈ messages_p | r ≠ r_p}
```

Figure 1 Message management for process p during single-shot Tenderbake. In red the new lines added with respect to Tenderbake [1].

5.4.2 Correctness proofs

▶ **Lemma 1.** *Let b_1 and b_2 two decided blocks, if b_1 and b_2 are in the same fork, then* detection(b_1, b_2) *returns at least $T + 1$ processes accountable as faulty.*

Proof. First, consider that b_1 and b_2 are two blocks in an Intra-round fork, i.e., those blocks are decided at the same round. In this case Algorithm in Figure 3 returns the intersection of the endorsement quorums (line 64). Since $2T + 1$ distinct signatures compose each QC and correct processes sing just once per round, then the intersection of two quorums contains at least $T + 1$ distinct processes.

Now, let us consider that we have a Cross-round fork composed of two blocks decided during different rounds. Let us assume, w.l.o.g., that b_1 is the block associated with the smallest round. In this case Algorithm in Figure 3 returns as accountable the intersection of the quorum of endorsement messages from b_1 and the refinement (line 65) of b_2 relatively to the round of b_1 (line 60). All we need to show is that: the refinement also returns a quorum of processes; hence, the intersection with the QC of b_1 contains at least $T + 1$ processes.

```
28   PROPOSE phase:
29    if proposer(ℓ_p, r_p) = p then
30      u := if endorsableValue_p ≠ ⊥ then endorsableValue_p
31           else newValue()
32      payload := (headCertificates_p, u,
33                  endorsableRound_p, preendorsementQC_p)
34      broadcast Propose_p(ℓ_p, r_p, h_p, payload)
35    handleEvents()

36   PREENDORSE phase:
37    if ∃q, eQC, u, eR, pQC :
38       Propose_q(ℓ_p, r_p, h_p, (eQC, u, eR, pQC)) ∈ messages_p ∧
39       (lockedValue_p = u ∨ lockedRound_p ≤ eR < r_p) then
40       if ∃(LR_p, LV_p, ∅) in justification_p then
41         replace (LR_p, LV_p, ∅) in justification_p by (LR_p, LV_p, pQC)
42      broadcast Preendorse_p(ℓ_p, r_p, h_p, hash(u), justification_p.peQC)
43    else if lockedValue_p ≠ ⊥ then
44      broadcast Preendorsements(ℓ_p, r_p, h_p, preendorsementQC_p)
45    handleEvents()

46     ENDORSE phase:
47    if |preendorsements()| ≥ 2f + 1 then
48     if ∃(LR_p, LV_p, ∅) in justification_p then
49         replace (LR_p, LV_p, ∅) in justification_p by (LR_p, LV_p, preendorsements())
50      u := proposedValue()
51      lockedValue_p := u;  lockedRound_p := r_p
52      add (lockedRound_p, lockedValue_p, ∅) to justification_p
53      broadcast Endorse_p(ℓ_p, r_p, h_p, hash(u), justification_p.peQC)
54      broadcast preendorsementQC_p
55    handleEvents()
56    advance(getDecision())
```

Figure 2 Piggyback version of Tenderbake for process p. In red, the new lines added with respect to Tenderbake [1].

Hence, let us show that the refinement procedure returns a quorum (line 65). The procedure starts with a quorum of endorsements and replaces processes with their justification (as long as the justification is associated with a greater round than b_1). Hence, we replace a process in a quorum with a quorum of processes forming the justification. This ensures that we still possess a quorum after each modification and, therefore, that at the end, the refinement procedure returns a quorum. Consequently, the intersection of the quorum returned by the refinement procedure with the quorum of endorsement messages returns at least $T + 1$ distinct processes. ◀

In the next Lemma we show that the returned processes by Algorithm in Figure 3 are never correct, hence those are Byzantine.

▶ **Lemma 2.** *Given two blocks b_1 and b_2 being a fork, then* detection(b_1, b_2) *never returns a correct process.*

```
58 proc detection(b_1, b_2)
59   if sign(b_1).round < sign(b_2).round
60     return sign(b_1).members ∩ refinement(sign(b_2).members, sign(b_1).round)
61   else if sign(b_2).round < sign(b_1).round
62     return sign(b_2).members ∩ refinement(sign(b_1).members, sign(b_2).round)
63   else
64     return sign(b_1).members ∩ sign(b_2).members

65 proc refinement(QC, round)
66   if ∃e ∈ QC, ∃q ∈ justification(e), round(q) > round
67     return q such that ∃e ∈ QC, q ∈ justification(e), round(q) > round∧
68                ∀e ∈ QC, ∀q' ∈ justification(e), round(q') ≥ round(q) ∨ round(q') ≤ round
69   else return QC
```

Figure 3 Fork Accountability module at process p.

The intuition of the following proof in the case of CR forks is the following. Let us observe that the refinement procedure takes as input the QC associated to b_2 and r_1 the decision round of b_1. It returns the smallest justification present in messages in b_2 QC associated with a round $r > r_1$ if any, and returns QC associated with b_2 otherwise. Intuitively, a process that endorsed a block cannot preendorse or endorse for later blocks without having a valid justification attached to its message (which differentiates a flip-flopping from a faulty flip-flopping). Therefore, such a process should never be returned by the refinement procedure. Indeed, a correct process being always justified cannot be in the smallest justification as it itself has a smaller justification attached to itself.

Proof. Let us also start with the more straightforward case of an Intra-round fork. In this case, Algorithm in Figure 3 returns the intersection of two endorsement quorums for the same round. It implies that we return processes that sent two distinct endorsement messages for the same round, necessarily a byzantine failure. A single endorsement message can be sent per round.

Now let us look at the case in which we have a Cross-round fork. Let us assume, w.l.o.g., that b_1 is the block associated with the smallest round. Assume now by contradiction that a correct process p is returned in the intersection. That is p endorsed for b_1 and was returned in the refinement of b_2 relatively to the round of b_1.

Since p endorsed for b_1 it must have added a justification item (that can be empty) for the round and value of b_1 in its justification set (line 52). From the protocol, we see that p can only send preendorsement or endorsement messages that are justified (with a non-empty set) by a quorum that does not include itself and is associated with a round greater r_1. Indeed, if p sends a prendorsements, then either it already has a justification or it adds the preendorsement quorum from the proposer that is associated with a round greater than r_1 (line 41). Similarly, if p sends an endorsement message then it adds the set of received preendorsements justifying its vote to its justification set if not already justified (line 49). Therefore, p cannot be returned by the refinement procedure of b_2 relatively to the round of b_1 as it always has a justification attached to it – A contradiction. ◀

Combining the two preceding Lemmas we obtain the following Theorem about the completeness and accuracy of our accountability detector:

Table 1 Comparison of the different presented approaches.

Approach	IR Fork	FBIR Fork	CR Fork	FBCR Fork	Extra message space complexity
Tederbake	Yes	Yes	No	No	No extra costs
Tenderbake fully just.	Yes	Yes	Yes	Yes	Unbounded msg space and local memory
Tenderbake over rb	Yes	Yes	No	No	Unbounded local memory

▶ **Theorem 3.** *For each fork at least $T + 1$ committee members are accountable as faulty and no correct committee member is ever accountable as faulty.*

As we discussed, the solution is quite communication-intensive. So far, it does not seem that it can be much improved without revisiting Tenderbake more deeply. A possible idea was to keep these justifications information locally at each process. The problem is that there is a need for synchronous assumptions to provide accountability for forks. Indeed, when the fork is observed, then processes must be able to provide their justification within a known delay to be ensured not to be wrongly suspected. In that case, since τ is unknown, it is not possible to distinguish a slow correct process from a byzantine process withholding its non-adequate justifications.

5.5 Accountability with Tenderbake over reliable broadcast

The previous approach suffers from unbounded size messages, which is necessary if we want to perform accurate and complete accuracy despite the unreliability of the communication mean. In this section, we investigate the advantages of leveraging reliable communications. Indeed, in this context, there is no need for messages to carry unbounded justifications. We consider light piggyback justification with parameterizable depth d. Committee members justify the last d pre-endorsement issued after the d previous endorsement (if any). The justification is a set of $2T + 1$ pre-endorsement that allowed them to flip-flopping. Those justifications are further carried by the *peQC* associated with each endorsement message.

In this case, we can detect Byzantine processes in a few cases:

- IR-Fork and FBIR-Fork: always.
- CR-fork: only if a process endorses for a value at round r for a block b_1 and pre-endorses for a value $b_2 \neq b_1$ at round $r + d$.

If $d = \infty$ then we boil down in the previous approach.

The main limitation of this approach is that even though we eventually receive all messages observed by correct processes, in case of missing justification for a flip-flopping, we do not know if those messages will arrive (exonerating the baker) or they do not exist at all (incriminating the baker).

5.6 Discussion

Table 1 depicts the different Fork Accountability approaches. The first four columns refer to the four kinds of forks that can occur with Tenderbake when more than T Byzantine are present in the committee: Intra-Round Forks; Fully Byzantine Intra-Round Forks; Cross-Round Forks; and Fully Byzantine Cross-Rond Forks. For each of those fork kinds, we

state if we are able to have Fork Accountability with the given approach. Finally, the last column presents the accountability solution's complexity to the Tenderbake complexity. Before digging into that, let us recall that if we have less than T Byzantine committee members, then we know that, after τ, the consensus instance terminates in $f+2$ rounds [1]. Contrarily, termination property can be violated, Byzantine committee members can let the consensus run endlessly or end it with a fork at their will (violating the agreement property). Thus, there is no upper bound on the consensus instance duration. Indeed, the number of times a correct committee member can perform a flip-flopping is unbounded, which explains why Tenderbake with full justifications incurs the cost of unbounded message space and, consequently, unbounded local memory. However, if we consider that messages do not carry the total amount of justifications (thus, message space complexity is bounded), we still have that the total amount of messages is unbounded (due to the computation duration unbounded). It follows that Tenderbake with weak justifications, even though it gets rid of the unbounded message space, it still suffers the extra cost of the unbounded local memory. Unsurprisingly, reducing the message space complexity comes at the price of reducing the accountability capability. As a result, we obtain the same results as with Tenderbake unmodified, which is, on the light side, free of any extra costs. To complete this discussion, let us consider the Streamlet [9] case. As shown in [4], with few modifications to the protocol (in the way blocks are decided, with no impact on the message and space complexity), we obtain an Accountable protocol. In a nutshell, differently than Tenderbake, processes when observing that a new proposed block b collects a quorum of pre-endorsement messages, rather than locally locking on it, they add such block to their local structure (a block-tree rather than a chain). In that case, we say that b is notarized, which is different than decided. After, new blocks are proposed. When b is followed by other notarized blocks such that it meets a particular condition, b is decided. Intuitively, in Streamlet the lock mechanism is explicit on chain, so there is no need to add extra information to the blocks as in Tenderbake, and that would also be true if with lossy channels before τ (Indeed, after τ correct processes would successfully synchronize with other processes being able to retreive the missed notarized and decided blocks, yet losing the messages exchanged to get to the notarization of those blocks). For such a reason is, Streamelet we can perform Fork Accountability with no extra costs because those costs are already paid by the protocol as it is in terms of space occupied by the blockchain data structure.

6 Conclusion and Future Work

This work lays the groundwork for Fork Accountability solutions for BFT-like consensus algorithms, as Tenderbake. Even though the proposed solutions are either not fully solving the Fork Accountability problem or are impractical, we believe that is an essential step toward better understanding the Accountability possibilities and impossibilities in the Blockchain context. In future work, we aim to explore how to increase Accountability capabilities practically. A possible direction to explore is the redefinition of the BFT-Consensus algorithms with bounded buffers in the accountability lens.

References

1 Lăcrămioara Aştefănoaei, Pierre Chambart, Antonella Del Pozzo, Thibault Rieutord, Sara Tucci-Piergiovanni, and Eugen Zălinescu. Tenderbake - A Solution to Dynamic Repeated Consensus for Blockchains. In *4th International Symposium on Foundations and Applications of Blockchain 2021 (FAB 2021)*, pages 1:1–1:23, 2021.

2 Amitanand S Aiyer, Lorenzo Alvisi, Allen Clement, Mike Dahlin, Jean-Philippe Martin, and Carl Porth. Bar fault tolerance for cooperative services. In *Proceedings of the twentieth ACM symposium on Operating systems principles*, pages 45–58, 2005.
3 Victor Allombert, Mathias Bourgoin, and Julien Tesson. Introduction to the Tezos Blockchain. In *Proc. High Performance Computing and Simulation*, 2019.
4 Emmanuelle Anceaume, Antonella Del Pozzo, Thibault Rieutord, and Sara Tucci-Piergiovanni. On Finality in Blockchains. In *25th International Conference on Principles of Distributed Systems (OPODIS 2021)*, pages 6:1–6:19, 2022.
5 Roberto Baldoni, Jean-Michel Hélary, and Sara Tucci Piergiovanni. A methodology to design arbitrary failure detectors for distributed protocols. *J. Syst. Archit.*, 54(7):619–637, 2008.
6 Ethan Buchman, Jae Kwon, and Zarko Milosevic. The latest gossip on BFT consensus. *CoRR*, 2018.
7 Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget. *CoRR*, 2017.
8 Miguel Castro and Barbara Liskov. Practical Byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 2002.
9 Benjamin Y Chan and Elaine Shi. Streamlet: Textbook streamlined blockchains, 2020.
10 Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM (JACM)*, 43(2):225–267, 1996.
11 Jing Chen and Silvio Micali. Algorand: A secure and efficient distributed ledger. *Theor. Comput. Sci.*, 2019.
12 Pierre Civit, Seth Gilbert, and Vincent Gramoli. Polygraph: Accountable byzantine agreement. In *IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*, 2021.
13 Pierre Civit, Seth Gilbert, Vincent Gramoli, Rachid Guerraoui, and Jovan Komatovic. As easy as abc: Optimal (a) ccountable (b) yzantine (c) onsensus is easy! In *36th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2022)*, 2022.
14 J. A. Garay, A. Kiayias, and N. Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Proc. EUROCRYPT International Conference*, 2015.
15 L.M. Goodman. Tezos – a self-amending crypto-ledger, 2014.
16 Andreas Haeberlen, Petr Kouznetsov, and Peter Druschel. Peerreview: practical accountability for distributed systems. In Thomas C. Bressoud and M. Frans Kaashoek, editors, *Proceedings of the 21st ACM Symposium on Operating Systems Principles (SOSP 2007)*, 2007.
17 Jae Kwon and Ethan Buchman. Tendermint.
18 S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System, 2008.
19 Peiyao Sheng, Gerui Wang, Kartik Nayak, Sreeram Kannan, and Pramod Viswanath. Bft protocol forensics. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 1722–1743, 2021.
20 Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.
21 Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. HotStuff: BFT consensus with linearity and responsiveness. In *Proc. ACM Symposium on Principles of Distributed Computing*, 2019.

A Appendix – CR fork Scenarios

We describe two detailed scenarios dealing with the (faulty) flip-flopping case to justify the need to modify Tenderbake with justifications and their re-transmissions.

A.1 Scenario 1

In this first scenario, we consider a system composed of 7 processes, $p_0, \ldots, p_6$, among which p_4, p_5 and p_6 are byzantine. The execution comprises three rounds of communication and leads to a fork with the accountability of faulty processes.

Figure 4 depicts the first scenario. In the first round, p_0 proposes A (p A message at Round 1, phase p in Figure 4) and all processes pre-endorse A (pe A message at Round 1, phase pe in Figure 4) but only processes p_1 and p_2 endorse A among the correct processes (e A message at Round 1 represented by the lock image, phase e in Figure 4). As byzantine processes also endorse A, the block A is created with a Quorum Certificate composed by the endorsement messages from p_1, p_2, p_4, p_5 and p_6, but all such endorsements are received only by Byzantine processes (down-going black arrows Round 1, phase e in Figure 4) that withhold the block just created.

In the second round, p_3, which did not see the block A created, proposes a new block B. p_0 and p_3 pre-endorse it as they are not locked on A, along with byzantine processes that do not follow their lock. This set of pre-endorsements (that fulfills a quorum) is received by p_0 and p_2. The rounds end without receiving enough endorsements to form a quorum certificate.

In the last round, p_4 proposes B. This time B (being not new) has a valid round associated greater than 1, the round in which processes locked for A. Hence, p_2 can pre-endorse it along with p_0 (which is not locked) and all byzantine processes. This leads to a pre-endorsement quorum certificate that is received by p_1 and p_2. The round terminates by the endorsements of p_1 and p_2 along with the three endorsements for B by the byzantine processes, p_4, p_5, p_6, the same quorum certificate of block A. It leads to a fork once both blocks are diffused.

Figure 4 Scenario 1. We refer to the propose, pre-endorse and endorse phase with p, pe and e respectively. We refer to the proposal (pre-endorse) message for a value X with p (pe) X notation.

In this scenario, we can see that the byzantine misbehavior that led to the fork is that byzantine processes first endorsed for A at round 1 and pre-endorsed B just after, in round 2. It is highlighted in Figure 5. Indeed, a correct process locked on block A during round 1 would refuse to pre-endorse B in a later round $1 + k, k > 0$, unless it observes a quorum of pre-endorsement for B in a round $1 < r < 1 + k$. In this particular case, any correct process would never obtain a valid pre-endorsement quorum for B before round 2 to justify the pre-endorsement for B while locked on A.

Scenario 1

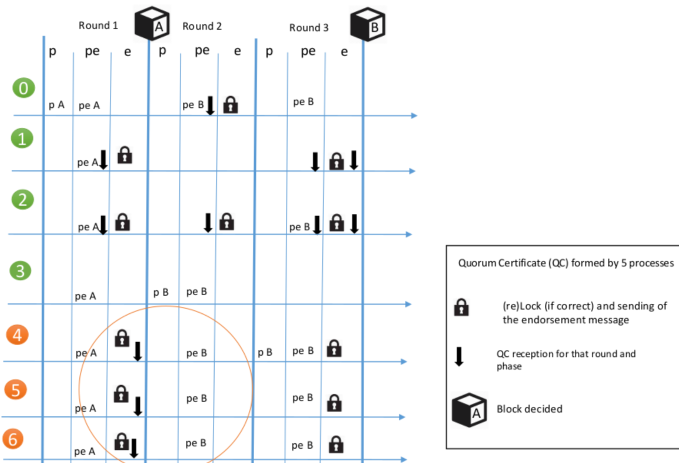

Figure 5 Example of how to detect Byzantine processes given a complete knowledge on the exchanged messages. In that case, p_4, p_5 and p_6 performs a faulty flip-flopping.

We consider a slightly modified version of Tenderbake. Processes produce justifications when locked on a value v and pre-endorse (endorse) a different value $v' \neq v$. That is, justifying their action by adding the pre-endorsement quorum certificate ($peQC$) for v' when sending a pre-endorsement message in the flip-flopping context. Note that endorsements carry all justifications issued during all rounds since the first lock round. Since those justifications chain are carried by endorsement messages then we collect them in decided blocks, that is, from processes which endorsed the block itself, p_1, p_2, p_4, p_5 and p_6 in this particular Scenario. Hence in this case, given the blocks A and B we collect information from correct processes p_1 and p_2 (Byzantine processes can omit justifications in their endorsement message).

- block A decided at round 1: such block comes with eQC, a quorum certificate of endorsements from p_1, p_2, p_4, p_5 and p_6 (first red rectangle in Figure 6), and none of them has justifications .
- block B decided at round 3: such block comes with a QC of endorsements from p_1, p_2, p_4, p_5 and p_6 (fourth red rectangle in Figure 6) and with the following justifications:
  - p_2 justifies its endorsement with $peQC$ for value B at rounds 2 (second red rectangle in Figure 6) and 3 (third red rectangle in Figure 6). Indeed, p_2 witnessed both the endorsement QC for block B in round 2 and round 3.

In this scenario, byzantine processes can be detected as their endorsement for A in round 1 (available from eQC for A round 1 in block A) implies that they should have set a lock for A at round 1. This, along with the pre-endorsement by the byzantine processes in round 2 (available from $peQC$ from B from round 2 and 3), implies a violation of the lock mechanism. Hence, we can detect $T + 1$ byzantine processes by simply gathering information available to the processes in the normal execution of Tenderbake once the blocks originating a fork are collected.

A.2 Scenario 2

In the following scenario, we first show that the previous scenario's accountability approach results in the violation of completeness and accuracy properties. We further show how to modify the accountability approach such that we are still able to detect and account for $T + 1$ Byzantine processes once a fork occurs.

Scenario 1

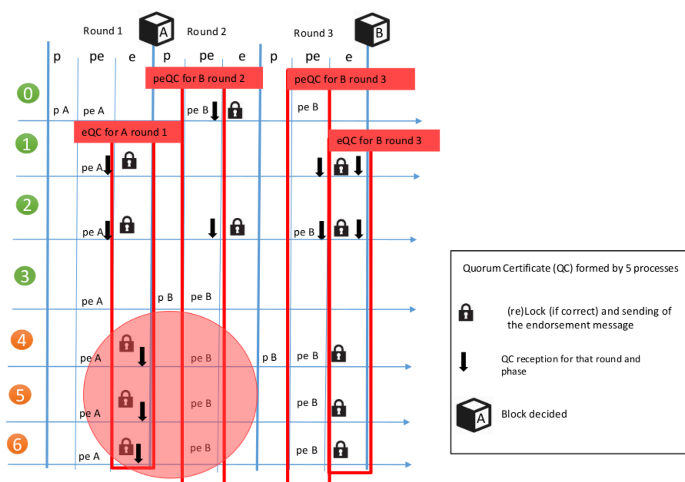

Figure 6 Justification gathered in decided block depicted using red rectangles along with highlight of detected byzantine process misbehaviour in light red.

The second scenario goes as follows and is depicted in Figure 7. During the first two rounds, we have the same execution as in Scenario 1. At the end of round 1, a block A is created with a Quorum Certificate composed of the endorsement messages from p_1, p_2, p_4, p_5, and p_6, but all such endorsements are received only by Byzantine processes that withhold the block just created. In the second round, p_3, which did not see the block A created, proposes a new block B. p_0 and p_3 pre-endorse it as they are not locked on A, along with byzantine processes that do not follow their lock. This set of pre-endorsements (that fulfills a quorum) is received by p_0 and p_2. The rounds end without receiving enough endorsements to form a quorum certificate.

Here, in the third and last round, we build the difference with the first scenario. p_4 proposes B as in Scenario 1. B has a valid round associated that is greater than 1 the round in which processes locked for A. Hence, p_2 can pre-endorse it along with p_0 (which is not locked at all) and all byzantine processes. This leads to a pre-endorsement quorum certificate that is received by p_1 and p_3 this time (in the first scenario it was p_2). The round terminates by the endorsements of p_1 and p_3 along with the three endorsements for B by the byzantine processes, p_4, p_5, p_6, a different quorum certificate of block A. It leads to a fork once both blocks are diffused.

If we use the same approach for the justifications as in Scenario 1, then we collect the information highlighted in Figure 8 and we might accuse p_2 of being Byzantine. Indeed, we get his endorsement at round 1 for A and his pre-endorsement for B at round 3, but not the justification for such pre-endorsement, that might look like a faulty flip-flopping. The same reasoning can be applied for the Byzantine processes p_4, p_5, p_6.

To solve this issue, we further modify Tenderbake. Processes produce justifications when locked on a value v and pre-endorse (endorse) a different value $v' \neq v$. This is done by adding the pre-endorsement quorum certificate ($peQC$) for v' when sending a pre-endorsement message in the flip-flopping context. Moreover, a process re-transmits all $peQC$ collected as justifications carried by pre-endorsement messages to other processes. That is, when sending a pre-endorsement message, it sends all justifications gathered so far. In this case,

Scenario 2

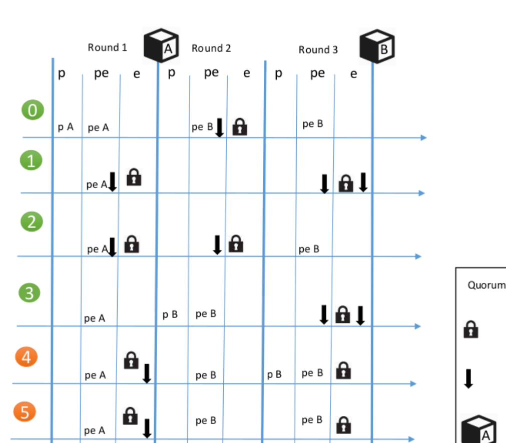

Figure 7 Scenario 2 representation. In particular we refer to the propose, pre-endorse and endorse phase with p, pe and e respectively. Moreover, when we refer to the proposal (pre-endorse) message for a value X with p (pe) X notation.

Scenario 2

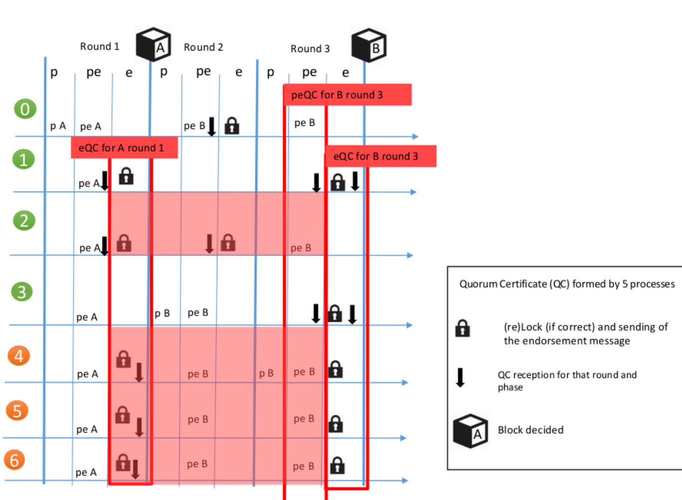

Figure 8 p_4, p_5 and p_6 performs a faulty flip-flopping, but their behaviour is indistinguishable from the p_2 given the subset of messages observed.

any endorsement message carries all justifications issued and collected following the same logic applied for the pre-endorsement messages justifications. Since those justifications chain are carried by endorsement messages then we collect them in decided blocks, that is, from processes which endorsed the block itself, p_1, p_3, p_4, p_5 and p_6 in this particular Scenario. Hence in this case, given the blocks A and B we collect information from correct processes p_1 and p_3 (Byzantine processes can omit justifications in their endorsement message).

- block A decided at round 1: such block comes with eQC from p_1, p_2, p_4, p_5 and p_6 (first red rectangle in Figure 6), and none of them has justifications.
- block B decided at round 3: such block comes with a eQC from p_1, p_3, p_4, p_5 and p_6 (fourth red rectangle in Figure 6) and with the following justifications:
  - p_1 and p_3 justifies directly its endorsement with $peQC$ for value B at rounds 3 (third red rectangle in Figure 9). Indeed, p_1 and p_3 witnessed eQC for block B in round 3.
  - p_1 and p_3 carries the justification from p_2, that is the pre-endorsement quorum certificate, $peQC$, for value B at round 2 (second red rectangle in Figure 9). Indeed, p_2 witnessed the endorsement QC for block B in round 2 and is part of the pre-endorsement quorum certificate for tound 3.

In this way, we have enough information to distinguish and account for the fork $T + 1$ Byzantine processes in the same way we did in the Scenario 1 context.

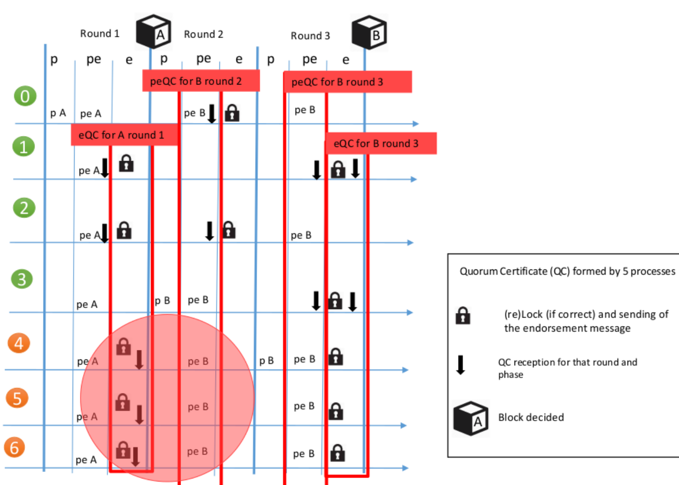

Figure 9 Justification gathered in decided block depicted using red rectangles along with highlight of detected byzantine process misbehaviour in light red.

Dynamic Blockchain Sharding

Deepal Tennakoon
University of Sydney, Australia

Vincent Gramoli
University of Sydney, Australia

Abstract

By supporting decentralized applications (DApps), modern blockchains have become the technology of choice for the Web3, a decentralized way for people to interact with each other. As the popularity of DApps is growing, the challenge is now to allocate shard or subnetwork resources to face the associated demand of individual DApps. Unfortunately, most sharding proposals are inherently static as they cannot be adjusted at runtime. Given that blockchains are expected to run for years without interruption, these proposals are insufficient to cope with the upcoming demand.

In this paper, we present dynamic blockchain sharding, a new way to create and close shards on-demand, and adjust their size at runtime without requiring to hard fork (i.e., creating duplicated instances of the same blockchain). The novel idea is to reconfigure sharding through dedicated smart contract invocations: not only does it strengthen the security of the sharding reconfiguration, it also makes it inherently transparent as any other blockchain data. Similarly to classic sharding, our protocol relies on randomness to cope with shard-takeover attacks and on rotating nodes to cope with the bribery of a slowly-adaptive adversary. By contrast, however, our protocol is ideally suited for open networks as it does not require fully synchronous communications. To demonstrate its efficiency, we deploy it in 10 countries over 5 continents and demonstrate that its performance increases quasi-linearly with the number of shards as it reaches close to 14,000 TPS on only 8 shards.

2012 ACM Subject Classification Computing methodologies → Distributed algorithms

Keywords and phrases Reconfiguration, smart contract, transparency, shard

Digital Object Identifier 10.4230/OASIcs.FAB.2022.6

Funding This research is supported under Australian Research Council Future Fellowship funding scheme (project number 180100496) entitled “The Red Belly Blockchain: A Scalable Blockchain for Internet of Things”.
Vincent Gramoli: Australian Research Council

1 Introduction

Blockchains, which originally aimed at enabling transparent asset transfers between permissionless individuals [23], has become the de-facto technology for the new version of the World Wide Web, called Web3. In January 2022 alone, the total volume of Web3 sales through decentralized applications (DApps) represented $16B [19]. These DApps are an appealing alternative to centralized applications, because they offer a **transparent** execution on **secure** data. Unfortunately, DApps create congestions on popular smart contract blockchains, like Ethereum [32]. The key idea to reduce this congestion is called *sharding*, which consists of splitting the workload across disjoint set of computers called shards, subnetworks or zones. In the context of DApps, sharding typically means executing distinct sets of DApps or smart contract functions on different sets of computers [14].

Unfortunately, the existing blockchain sharding protocols (Table 1, later detailed in Section 5) suffer from limitations. In fact, they are typically *static*: once the blockchain is spawned, there is no way to change the number of shards it uses. The problem is that blockchains are intended to run for a long time (e.g., Bitcoin [23] has been running for more than a decade without interruption) whereas new DApps are continuously uploaded to

© Deepal Tennakoon and Vincent Gramoli;
licensed under Creative Commons License CC-BY 4.0
5th International Symposium on Foundations and Applications of Blockchain 2022 (FAB 2022).
Editors: Sara Tucci-Piergiovanni and Natacha Crooks; Article No. 6; pp. 6:1–6:17
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Table 1 Comparison of sharded blockchains: the dynamism ranges from low, as indicated by ○, to high, as indicated by ●, a checkmark ✓ indicates that the property holds while a cross ✗ indicates that that the property does not hold and a dash "–" indicates that it remains unknown.

	Sharded blockchains	Transparency	Dynamism	Shard number dynamism	Shard size dynamism	No synchrony needed
Payments	Elastico [22]	✗	◔	○	◑	✗
	OmniLedger [20]	✗	◑	○	●	✗
	RapidChain [33]	✗	◑	○	●	✗
	SSChain [6]	✗	◑	○	●	✗
DApps	Avalanche [27]	✗	●	●	●	✗
	ChainSpace [1]	✓	◑	○	●	✓
	Cosmos [21]	✗	◕	●	◑	✓
	Eth2 [31]	–	◔	○	◑	✓
	Polkadot [5]	✗	○	○	○	✓
	Zilliqa [29]	✗	○	○	○	✗
	This work	✓	●	●	●	✓

blockchains at runtime. The popularity of these DApps is heterogeneous and a new popular DApp may severly increase the number of requests to a particular smart contract, just like the CryptoKitties DApp that congested the Ethereum network [16]. Ideally, a sharding protocol should allow the blockchain governance to resize the shards and adjust the shard number on-demand without *hard forking*, i.e., creating a duplicated instance of the same blockchain. This would allow to seemlessly migrate DApps from one shard to a newly created one, hence provisioning more resources for popular DApps, grouping less demanded DApps on fewer shards, or offering more resources (e.g., CPU, storage) to a particularly congested shard.

Another problem is that most sharding protocols are *opaque* (cf. 2nd column of Table 1): there is no way to securely access their shard configuration. Even if a sharding protocol was made dynamic by offering the users to change the number of shards at runtime, there would be no secure way for these users to confirm the changes took effect. In some cases, sharded blockchains offer a website where users can find information about the current shard configuration. For example, Cosmos [21] offers a website to observe a map of its zones [25]. However, such a web service is typically centralized and prone to a single point of failure, hence defeating the purpose of using a distributed ledger for security. First, this website could simply be hacked, conveying a misleading sharding configuration. Second, the traffic towards the website could be easily redirected with a network attack [12]. Finally, users could expose themselves to phishing attacks by accessing a hacked copy of the website instead of the real one. Such attacks are becoming frequent to fool blockchain users about the information they access online [4].

In this paper, we propose a new dynamic blockchain sharding protocol. As it is intended to operate in open networks, it does not assume synchrony but partial synchrony [11], in that the bound on the message delays is unknown. This protocol is made *transparent* by exploiting the blockchain itself: a minimum number of users can (i) create, (ii) close or (iii) adjust the size of a shard by invoking functions of a smart contract residing on the default shard (called *mainchain*) within a limited time window (if the network asynchrony prevents them from succeeding, then they retry with a larger time window until success).

As all smart contract invocations are logged to the secure storage of the distributed ledger, the shard configuration is securely visible from the world state. Similar to Eth2 [14], a new shard is created as a *shard chain* provisioned by the assets deposited on the mainchain by its users. The most important challenge we had to solve was for the network topology to adapt based on the output of the sharding smart contract: the reconfiguration function emits an event that triggers the spawning, shutdown and restart of some of the blockchain machines.

Like other sharding approaches we provide randomness in shard creation to prevent shard takeovers by malicious nodes. We also present a shard committee rotation approach to mitigate bribery by a slowly-adaptive adversary and a mapping of transactions to shards. Finally, we evaluate our solution on a scalable blockchain called CollaChain [30], which combines DBFT [7], a formally verified [3] consensus protocol, that makes CollaChain fork free; and a Scalable version of the Ethereum Virtual Machine, called SEVM, making it compatible with the largest ecosystem of DApps. Our results confirm that our sharding protocol leads to quasi-linear speedup, that the performance of shards can benefit from a growing number of node resources, and that our mainchain does not act as a performance bottleneck. To summarize, our contribution is threefold:

- We introduce dynamic blockchain sharding, the ability for a blockchain to reconfigure the number of its shards and the size of each of its shards without disrupting the blockchain service. This ability is particularly appealing to cope with the growing of DApps over recent blockchains.
- We propose a dynamic sharding solution that creates a new shard, adjusts a shard, closes a shard, and rotates the shard participants. We implement these algorithms as inherently transparent smart contracts that emit events to replace the current sharding configuration at the network level.
- We demonstrate the feasibility of our approach by implementing our algorithms within a recent scalable blockchain that we deploy in 10 countries across all 5 continents. The experimental results confirm that the performance scales quasi-linearly with the number of shards and demonstrate that the system can achieve close to 14,000 TPS with only 8 shards.

The rest of this paper is ordered as follows: In Section 2 we provide the model and preliminary definitions. In Section 3, we present our dynamic sharding protocol. In Section 4, we illustrate the performance of our solution when deployed at large scale. In Section 5, we discuss the related work. In Section 6, we conclude.

2 Preliminaries

2.1 Blockchain

A blockchain is a decentralized, distributed system that processes user transactions and logs the transactions to an auditable and cryptographically secure ledger. Each participant keeps a replicated state of the system, and all validator/miner participants require to reach consensus to agree on the set of transactions to be executed. The agreed upon transactions reside in the body of a block data structure and each block has a pointer to the previous block building a chain of blocks known as the blockchain. The header of a block structure consists of a root of a merkel tree known as the state root. The *state root* represents the state of the blockchain at a specific block. Each participant at block N will have the same state root. By traversing through the Merkle tree from the state root in a block, the accounts, balances, contract data and contract state can be retrieved. A blockchain *committee* is a set

of blockchain participants that execute consensus separately from the rest of the network. In this work, each blockchain committee maintains its own state and process a separate set of transactions.

2.2 Model

In our system model, we assume our system consists of n participants. Each participant controls a SEVM and Consensus node in Collachain [30]. Participants join the network in a permission-less manner as outlined in Section 2.3 to tolerate Sybil attacks. From the sample of joined nodes, a group of nodes are selected through a random mechanism (Section 2.3) to the main chain. The set of nodes that execute consensus on the mainchain are termed the *mainchain committee*. Mainchain comittee is tasked with administrative tasks of the network such as shard creation, shard committee rotation, and dynamic adjustment of the number of shards and the nodes in a shard. A mainchain can create one or many shards from participants, we term as validator candidates. Each shard keeps separate state, and transactions and is tasked with executing a unique DApp. The shard committee (i.e. the set of validators in the shard) rotates per epoch which signifies a time t that is sufficiently small to avoid shard takeover by a slowly-adaptive adversary (Section 2.2).

Our network model assumes honest nodes in the network are well-connected and the communication channels between honest nodes are partially synchronous, i.e., if honest nodes broadcast a message, all honest nodes receive it after an unknown time T and a bounded maximum delay of δ. While various sharded blockchains typically assume a stronger property, called synchrony [11], where the upper bound on the delay of every message is known, note that synchrony is typically difficult to guarantee and can easily be violated in an open network like the Internet [12], which has led to numerous double spending attacks against blockchains [24, 13].

In our threat model, out of n participants, we assume f are byzantine such that $f < n/4$. This is to ensure that a committee has k nodes such that $f_k < k/3$ with high probability, similar to previous work [20, 22]. Note that n and k can vary at run-time due to the dynamism to our approach. Only 1/3 of the mainchain nodes can be byzantine. The byzantine nodes can behave arbitrarily or collude to attack the system. All correct nodes adhere to the presented protocol (Section 3). In order to cope with a *Sybil attack*, whereby an adversary forges fake identifies to outnumber significantly the participants identities, we use a proof-of-stake (PoS) mechanism as described in the subsequent section. A bribery consists of an adversary incentivizing a committee participant to join its coalition. A *shard-takeover attack* consists of an adversary gaining control over sufficiently many nodes within a committee to prevent consensus from being reached. As explained below we assume a slowly-adaptive adversary that can bribe all nodes but only progressively (not instantaneouly) [20, 33] and we cope with this attack by proposing a rotating committee.

2.3 Bootstrapping

We consider a *permissionless* model. Any participant can join or leave the network without permission. Our membership protocol is similar to Algorand [18] where participants that require to be a part of consensus needs to stake some coins. A weight is assigned to each joining participant based on their stake. Consequently, a subset of participants are selected to perform consensus on the mainchain based on a random beacon and the weights of the nodes. This helps prevent Sybil attacks. To mitigate bribery take-overs, the main chain committee rotates periodically and the size of the main chain is changeable in a similar

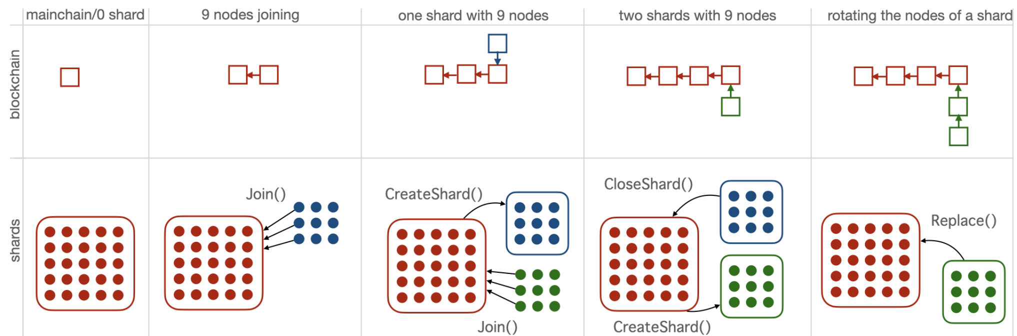

Figure 1 An example of the consecutive steps (from left to right) of a dynamic sharding execution where 2 shards are created, one of these shards is closed and shard nodes are rotated.

approach to how we change the shard size (Algorithm 1 line 15), which allows new nodes to join the main chain committee if a threshold of mainchain participants agree. We do not implement our membership protocol in this paper but leave it as a part of future work.

3 The Dynamic Sharding Protocol

In this section, we present how the dynamic sharding protocol adjusts the size and number of shards, and how it rotates shard nodes.

3.1 Overview

Figure 1 depicts a high level example of a dynamic blockchain sharding execution where smart contract invocations stored in blocks reconfigure the sharding. Initially, there are 25 participants in the mainchain with a single genesis block, as depicted on the 1st column, they decide the shard size. Then, external participants invoke the Join($\cdot$) function to join a new shard (cf. 2nd column). When enough of them have joined, the CreateShard($\cdot$) function is invoked on the mainchain smart contract and creates a new (blue) shard (cf. 3th column). The resulting function invocation is stored as a transaction of a new block of the mainchain. A new (blue) shard chain, maintained by the shard is created: it is linked to the block of the mainchain where its creation invocation is stored. New participants invokes the Join() function as depicted in the middle column. After that, the CreateShard($\cdot$) function creates a new (green) shard while the old (blue) shard invokes the CloseShard($\cdot$) function, which reports the blue shard history to a new block of the mainchain (cf. 4th column). Finally, the new shard rotates its participants by executing the Replace($\cdot$) function whose invocation gets stored in a new block.

3.2 Shard creation

The shard creation is presented in Algorithm 1 and is deployed on the mainchain as a smart contract during the bootstrap of the blockchain. The variable *admins* keeps track of a set of mainchain participants and *NumberOfAdmins* refers to the number of participants in the mainchain.

The shard creation smart contract is initialized with a set of data structures. The variable event refers to a broadcast message sent to all blockchain nodes in a shard. The event has a name (e.g., ShardNodes) and values that it broadcast (i.e. ShardNodes broadcasts an unsigned

Algorithm 1 The smart contract that triggers the creation of a new shard.

```
 1: Initialization:
 2:    event ShardNodes(uint, string[], address[])
 3:    uint shardSize
 4:    uint NumAccounts
 5:    mapping (string → string[])shard
 6:    mapping (address → bool))called
 7:    mapping (string → bool))voted
 8:    mapping (uint → address[])accounts
 9:    mapping (uint → uint)SizeOfShard
10:    mapping (uint → uint)NumberOfShards
11:    mapping (uint → bool)Created
12:    admins : the set of addresses of admins
13:    NumberOfAdmins = | admins |
14:
15: SetSize(val, NShards):            ▷ threshold of admins set shard size, number & accounts/shard
16:    if SenderAddr ∈ admins then                                  ▷ if function invoker is an admin
17:       SizeOfShard[val]++
18:       NumberOfShards[NShards]++
19:       if NumberOfShards[Shards] == (2 * NumberOfAdmins/3-1) &
              SizeOfShard[val] == (2 * NumberOfAdmins/3-1) then  ▷ if shard size, number, accounts agreed
20:          shardSize = val
21:          NumShards = NShards
22:
23: JoinShard(ipAddr):                               ▷ when a node wants to join a shard as validator
24:    if called[senderAddr] == false & voted[ipAddr] == false then   ▷ avoid assigning IP twice to shard
25:       called[senderAddr] = true
26:       voted[ipAddr] = true
27:       random = RANDContractaddr.GetRand()                   ▷ Fetch random number from RANDAO
28:       shard[random mod (NumShards)] ∪ ipAddr
29:       accounts[random mod NumShards] ∪ senderAddr
30:       if length(shard[random mod (NumShards)]) == shardSize & Created[tag] == false then
31:          CreateShard(random mod (NumShards),shard[random mod (NumShards)],
32:                accounts[random mod (NumShards))
33:          length(shard[random mod (NumShards)]) = 0                  ▷ reset the shard tag value to 0
34:          length(accounts[random mod (NumShards)]) = 0               ▷ reset the shard tag value to 0
35:
36: ClosedShards(tag):                ▷ admin calls this if received n-t COMMITS for close from shard chain
37:    Closed[tag] = true
38:
39: CreateShard(tag, []shard), []accounts):
40:    emit ShardNodes(tag, shard, accounts)   ▷ emit ip addresses & accounts of shard nodes, triggers shard start
41:    Created[tag] = true                                                  ▷ Assign shard as created
```

integer, a string array and an address array). A *mapping* is a data structure mapping a key to a value. The *bool* is a boolean data type and | *admins* | is the set containing the wallet addresses of admin nodes.

Admins of the main chain start by setting the size of shards and the number of shards (Algorithm 1 line 15). Note that a threshold of admins should agree to the same settings for these values to be set (Algorithm 1 line 19), and a threshold of admins can again agree to change these values during run time making the sharding dynamic. Note that unlike any other sharding scheme, we provide the capability to change shard size and number of shards even when the number of nodes in the network remains constant (no nodes joining or leaving).

The validator candidates invoke the JoinShard function and parse their IP address (Algorithm 1, line 23) in an attempt to join a shard. Note that, at line 24 of Algorithm 1, prevents validator candidates from joining multiple shards as well as two validator candidates from joining shards with the same IP address. Consequently, the shard creation

contract fetches a random number *random* from a verifiable random number generation contract (Algorithm 1, line 27) taken out of our system. (We rely here on the RANDAO implementation [26] of a random number generator that is expected to be used in Ethereum 2.0 as an example only. Randao is synchronous but a partially-synchronous random number generation solution can easily be used instead [9]).

Based on *random*, the IP address of a validator candidate is assigned to a random key of a *shard* mapping (Algorithm 1, line 28). Deriving the key values as: *random* mod *NumShards* ensures that the IP address of a candidate is assigned a shard tag x such that $x \in \{0, 1, ..., NumShards\}$.

Similarly, wallet address of the validator candidate is also added to a random key corresponding to a shard tag of an *account* mapping.

We underscore that *NumShards* can be adjusted by admins to accommodate more, or less shards in the system. If for a particular shard key/tag the maximum number of nodes (i.e., *ShardSize*) that could be assigned is complete, a CreateShard function is invoked, parsing an array of validator candidates and validator accounts for a shard tag (Algorithm 1, line 32).

The CreateShard function, emits a smart contract event *ShardNodes* (i.e., a broadcast to all participants) with the validator IP addresses and accounts that should be in a particular shard tag (Algorithm 1, line 32).

Validator candidates upon receiving *ShardNodes* event verifies its IP address is included in the event. If included, the validator candidates reconfigure and form a validator committee for a shard with a specific tag. Details of this process is outlined in Algorithm 2.

Algorithm 2 The algorithm executed by a participant to create a new shard upon reception of the smart contract creation event.

```
1: Upon receiving a smart contract event:
2:   event ← subscribe(CreateContractAddr)    ▷ all nodes subscribe to events from shard create smart contract
3:   if localIP ∈ event then                   ▷ If local IP is in event
4:     tag, shard, accounts ← extract(event)   ▷ extract values from event
5:     stop(node)
6:     editGenesis(accounts)                   ▷ Edit genesis with accounts
7:     connectPeers(shard)                     ▷ connect with other members of the shard
8:     start(node)
```

3.3 Shard closing

Shard closing is a procedure that helps prevent resource wastage. If a shard is not processing many transactions or is idle for a while a shard nodes can decide to close the shard. This is a part of the extended dynamism our protocol provides.

Algorithm 3 presents the smart contract algorithm for closing a blockchain shard in a partially synchronous manner. The variable N_v is the number of nodes that the shard contains.

Firstly, once a shard node decides to close the shard it is a part of, it invokes the CloseShard (Algorithm 3 line 10) parsing the state root the node prefers to close at. Algorithm 3 line 11-line 12, ignores if a state root is parsed to the function by participants more than once. Otherwise, the *threshold* is increased (Algorithm 3, line 14), which indicates the number of participants that have parsed a specific state root to the CloseShard function. At line 15 of Algorithm 3, if $2 \cdot N/3 + 1$ nodes s.t. N is the total number of participants in the shard have parsed the same state root to CloseShard, then a COMMIT event is emmitted with the *ShardTag*. Otherwise, the parsed state root to the CloseShard function is emitted in a *Bs* event.

Algorithm 3 The smart contract that triggers the closing of a new shard.

```
1:  Initialization::
2:    event Bs(string y)
3:    event COMMIT(string x, string m)
4:    mapping(string → uint) threshold
5:    uint Nv
6:    bool reached
7:    mapping(bytes32 → string) called;
8:    Nv = val                                  ▷ The number of nodes in a shard from Algorithm 1
9:    ShardTag = tag                            ▷ tag of shard generated– based on RANDAO in Algorithm 1
                                                ▷ nodes call CloseShard parsing the state root
10: CloseShard(stateroot):
11:   if called[hash(SenderAddr, stateroot)] == true then   ▷ SenderAddr parsed stateroot before
12:     return                                  ▷ avoids double voting
13:   called[hash(SenderAddr, stateroot)] = true             ▷ 'SenderAddr' parsed stateroot
14:   threshold[stateroot] = threshold[stateroot] + 1        ▷ number of nodes parsed specific 'stateroot'
15:   if threshold[stateroot] == 2*Nv/3+1 & !reached then   ▷ state root first reaching threshold
16:     reached = true
17:     emit COMMIT(stateroot, "COMMIT", ShardTag)           ▷ emits a commit event with the stateroot
18:   emit Bs(stateroot)                        ▷ emits event with the parsed stateroot
```

Algorithm 4 The algorithm executed by a participant to close a shard upon reception of the smart contract closure event.

```
1:  Upon receiving a smart contract event:
2:    event ← subscribe(CloseContractAddr)      ▷ all nodes subscribe to closing smart contract
3:    if contains(event, COMMIT) then           ▷ smart contract event contain the "COMMIT" string
4:      number = getCurrentBlockNumber()
5:      for i = 0; i < number; i++ do
6:        block ← getBlock(i)
7:        if block.stateroot = event.stateroot then
8:          Close(block.number)                 ▷ parse closing block number to sync balances algo
9:          exit()                              ▷ exit code
10:   else
11:     if nodeHas(event.stateroot) then        ▷ If node has same state root
12:       closeContractAddr.CloseShard(event.stateroot)     ▷ pass stateroot to SC
13:     else
14:       pending.push(event.stateroot)         ▷ push the stateroot to a pending array
15:       Check()

16: Check():                                    ▷ Do in parallel
17:   for i=0; i < length(pending); i++ do
18:     if nodehas(pending[i]) then
19:       CloseContractAddr.CloseShard(pending[i])          ▷ parse stateroot to smart contract
```

Algorithm 4 presents the execution at a shard participant when either a COMMIT or a *Bs* smart contract event is received from the shard close smart contract algorithm (Algorithm 3).

A shard participant subscribes to the close shard smart contract in its state. Upon receiving a smart contract event from this smart contract (*CloseContractAddr*) at the shard node, the *event* is filtered. Consequently, the shard participant checks if the event is a COMMIT event (i.e., whether it contains the keyword COMMIT) at Algorithm 4 line 3. If this condition is met, the current block number (Algorithm 4 line 4) of the participant is retrieved and the state of the shard participant is traversed from the 0th block to the current block to find the block number that contains the state root. If a block exists with the received state root in the shard node, it decides to parse the block number to a Close function (Algorithm 4 line 8) shown in Algorithm 5 and exits.

If the *event* is not of type COMMIT but the shard participant has the state root contained in the event (Algorithm 4 line 11), the participant invokes the CloseShard function in the Close shard smart contract parsing the state root. If the *event* is not of types COMMIT and

the shard node does not have the state root received, it is pushed to a pending array and kept (Algorithm 4 line 14), in case the shard participant sees the state root sometime in the future. In Algorithm 4 line 16, a *Check* function concurrently and repeatedly checks, if the shard node has the pending state root. The CloseShard function is invoked parsing the state root if the state root is found (Algorithm 4 line 19).

Algorithm 5 Shard chain participant Send Closing Account Balances to main chain.

```
1: Initialization:
2:    A is the set of account addresses in the shard
3:    Account(address, balance)                          ▷ A tuple of address and balance
4:    SA is the set of Account(address, balance) tuples

5: Close(BNumber):                    ▷ parse block number at which the shard should close
6:    for a ∈ A do
7:       b ← getBalance(a, BNumber)                     ▷ Balance of account a at closing block
8:       SA ∪ Account(a, b)
9:    Broadcast(SA, ShardTag)                            ▷ Broadcast to main chain nodes
10:   stop(shardNode)
```

Algorithm 5 executes at each shard participant and retrieves balances of all accounts at the block number that the shard closes (Algorithm 4 line 8) and broadcasts it and the shard tag to the main chain participants (Algorithm 5 line 9). Note that this broadcast is a reliable broadcast and waits for an ACK before the shard participants stop in the subsequent line.

Algorithm 6 Syncing of balance at the main chain from shard chains .

```
1: Initialization:
2:    threshold = 2N/3 + 1 s.t. N is the total number of shard chain nodes
3:    mapping (bytes32 → uint) count

4: Receive(SA), ShardTag:                                ▷ Receive Account tuple set
5:    count[hash(SA)] ← count[hash(SA)] + 1       ▷ times specific account tuple set received
6:    if count[hash(SA)] == threshold then               ▷ If threshold of same SA received
7:       CreateContractAddr.ClosedShard(ShardTag)
8:       stop(node)
9:       editGenesis(SA)            ▷ Edit the genesis, adding accounts and balances tuple set
10:      start(node)
```

A main chain participant upon receiving the tuple set of accounts and balances SA from shard participants, and the shard tag, executes Algorithm 6. Upon receiving SA, the algorithm keeps count of the number of unique SA sets received (Algorithm 6 line 5). If $2 \cdot N/3 + 1$ number of the same SA set is received s.t. N is the number of participants in the closing shard, the mainchain node invokes the *ClosedShard* function in Algorithm 1 to set the shard with the specific *tag* as closed. Consequently, the main chain participants stop (Algorithm 6 line 8), edits the genesis adding the new accounts and balances (Algorithm 6 line 9) and restarts (Algorithm 6 line 10). This way, the mainchain participants are synced with the accounts and balances of the shard chain. Note that, syncing accounts and balances from multiple shard chains upon shard closing does not affect the consistency of the state in the main chain since accounts in each shard are disjoint.

3.4 Shard committee rotation

A shard with a particular tag remains active once it is created until it is closed. There is a risk of participants being bribed by a slowly-adaptive adversary while a shard is active. If sufficiently many participants in a shard committee are bribed this way (at least 1/3), there is

a risk of shard takeover. To mitigate this risk, we propose a shard committee rotation protocol that is part of the shard creation smart contract (Algorithm 1) but presented separately below for clarity. We consider an epoch as a specific time t where a shard committee processes transactions. At every t interval, all correct shard participants performs committee rotation correctly. Note that the number of correct nodes in a shard is greater than $2N_v/3$.

Algorithm 1 Extension Shard committee rotation algorithm, a part of shard creation smart contract.

```
42: Initialization::
43:    mapping (string → uint) ReplaceIpInvoked
44:    mapping (address → uint) ReplaceAddressInvoked

45: Replace(ipAddr, tag):                                  ▷ Shard node parses its Ip address
46:    ReplaceIpInvoked[ipAddr]++
47:    ReplaceAddressInvoked[senderAddr]++
48:    if ReplaceIpInvoked[ipAddr] > 2 · N_v/3 & ReplaceAddressInvoked[senderAddr] > 2 · N_v/3 then
49:       called[senderAddr] = false
50:       voted[ipAddr] = false
51:       Created[tag] = false
52:       JoinShard(ipAddr)                                ▷ Invoke JoinShard in Algorithm 1
```

The committee rotation starts with shard participants invoking Replace in the Algorithm 1 Extension at line 45. Each correct shard participant parses the IP address of each of its committee members and their shard tag simultaneously to the Replace function. If a particular IP address and sender address has been used for the invocation $2 \cdot N_v/3$ times, the *called* and *voted* mappings are set to false for the corresponding IP address and sender address. Consequently, the *Create*[*tag*] is set to false and the JoinShard function is invoked parsing the IP address. The JoinShard function ensures shard participants are assigned to new shard committees following the same process of shard creation, that is, rotating the shards committees every epoch. Note at the end of an epoch before the committee is rotated, the mainchain participants can adjust the number of shards and the number of members per shard parameters according to the workload, which will change the number of shards and the nodes per shard, making our sharding approach dynamic.

3.5 Transaction assignment

In a web-scale blockchain that we foresee, each DApp executes on at most one shard. This concept is known as application or service-oriented sharding [17]. For example, we would have a Twitter DApp on one shard, a Youtube DApp on another shard. Each client sends requests to the shard that executes their required DApp. Each shard tag and the services they execute will be made available publicly so the clients can connect to the shard they prefer to send their transactions. Due to the service-oriented nature of sharding the state of each shard is disjoint, hence state consistency is not affected due to data migrations happening from shard closing and shard rotation of multiple shards. Also, due to the shard independence there is no need for cross-shard transactions. We do not present our own cross-sharding protocol and it is out of the scope of this paper.

3.6 Availability

Committee rotation in every epoch is essential to tolerate a slowly-adaptive adversary. However, frequent changes of committees is a challenge when the state is sharded. A new shard committee, needs to sync the state from a previous shard committee, to service the

DApps for a particular shard tag. This syncing process involves downloading the entire blockchain from previous nodes and is an expensive task, which is known to bottleneck performance and affect the availability of shard nodes for transaction processing [6]. Since our sharding approach is byzantine fault tolerant downloading the latest state would suffice by querying $f+1$ previous shard committee members. There is no need to download the entire state history (i.e. snapshots) nor the entire block history. As such, we provide better availability than some sharding approaches that shard the state as well [20].

3.7 Proof sketches

The first lemma shows that each shard contains less than $N_v/3$ byzantine participants with high probability. This is key to guarantee agreement among each shard to guarantee that the view of the blockchain is consistent across all replicas. For simplicity in the analysis, we assume that the shard participants correspond to a sample of N_v participants taken uniformly at random among the whole set of n participants and we reuse the same reasoning as in [22].

▶ **Lemma 1.** *In each shard of N_v participants, there are less than $N_v/3$ byzantine participants with high probability.*

Proof. By assumption we return a participant taken uniformly at random among all n participants. Consider each of this event as a Bernoulli trial such that a random variable X_i is 1 if the returned participant is correct and 0 if it is byzantine. Let ρ be the portion of byzantine participants. Because there are at most $f < n/4$ byzantine participants among the initial n participants, we have $\rho < 1/4$.

$$\Pr[X_i = 1] = \quad p = \frac{(n-\rho)}{n},$$
$$\Pr[X_i = 0] = \quad 1 - p = \frac{\rho}{n}.$$

The random variable $X = \sum_{i=1}^{N_v} X_i$ thus follows a binomial distribution and $\Pr[X = k] = \binom{N_v}{k}\rho^{N_v-k}(1-\rho)^k$, hence we can derive the probability $\Pr[X \leq 2N_v/3]$ of creating a shard with less than 2/3 of correct participants:

$$\Pr[X \leq 2N_v/3] \quad = \quad \sum_{k=0}^{2N_v/3} \binom{N_v}{k} \rho^{N_v-k}(1-\rho)^k.$$

As this probability decreases exponentially fast with N_v there exists a parameter λ and a constant n_0 for which $\Pr[X \leq 2N_v/3] \leq 2^{-\lambda}$ for all $N_v \geq n_0$. As a result, each shard contains at most $\lceil N_v/3 \rceil - 1$ byzantine participants with high probability, which concludes the proof. ◀

Given Lemma 1 and that our protocol relies on the DBFT [7] consensus protocol, which is resilient optimal, we know that participants agree when less than $N_v/3$ are byzantine. Hence each time a new block is added to a shard that did not fail, then the shard remains consistent with high probability. As a result, the transparent access to sharding information remains guaranteed. An important remark is that the proof of Lemma 1 relies on having $N_v \geq n_0$, however, for the sake of the empirical analysis we choose N_v relatively small (up to 60 machines) in Section 4 to limit the cost of our AWS experiments.

▶ **Lemma 2.** *If $2 \cdot N_v/3 + 1$ of the participants of shard s invoke its* CloseShard() *function with the same argument, then the shard s eventually closes.*

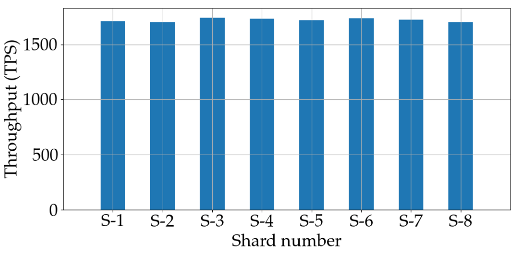

Figure 2 Throughput per shard.

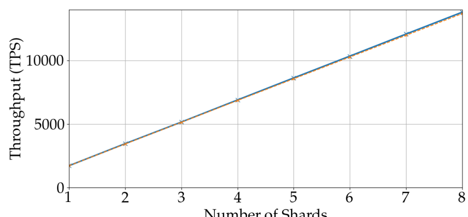

Figure 3 Linear increase in throughput with increasing number of shards.

Proof. The state root at which a shard participant wishes to close the shard is received by all correct participants in the shard by the Bs event (Algorithm 3, line 18) since the network is partially synchronous. Also, if a participant agrees to close the shard at a particular state root after seeing the state root event, they will either have that state root in their history or will eventually have it since consensus ensures the nodes have the same state history eventually. Therefore, if at some point in time, $2 \cdot N_v/3+1$ participants (Algorithm 3 line 15) – where the number of byzantine participants is $f < N_v/3$, agree on the state root, a commit event will be emitted (line 17) triggering the close of the shard. ◀

4 Evaluation

In this section, we evaluate the performance and the dynamism of our sharding solution. Our sharding approach was implemented on Collachain [30], a DApp supported blockchain. Note that while we evaluated on Collachain to benefit from the fork-free guarantees, our solution is adaptable for any Ethereum-based blockchain should the fork-free guarantees not be needed. We implemented our smart contract algorithms using Solidity and algorithms running at participant nodes using Web3js. All the experiments were performed on AWS, with c5.4xlarge (16 vCPUs, 32GB RAM) blockchain instances (i.e., which have similar performance to a modern PC) and c5.xlarge (4 vCPUs, 8GB RAM) client instances sending asset transfer transactions. A balanced workload was sent to each shard.

4.1 Dynamic shard adjustment

Figure 3 presents the scalability of our solution. Each shard consists of 60 machines evenly distributed across 10 AWS regions spanning 5 continents: Ohio, Mumbai, Seoul, Singapore, Sydney, Tokyo, Canada, Frankfurt, London, Paris, Stockholm, São Paulo. The dotted line is a straight line indicating the ideal speedup one could expect from multiplying the performance of the first shard by the number of shards. The continuous line represents the throughput with the number of shards. As we can see, the throughput increases almost linearly with the number of shards. At 8 shards, the throughput is 13,808 TPS. The quasi-linear growth in throughput is expected since, each shard processes a unique set of transactions, without performing cross-shard transactions because, as mentioned previously, our shards are dedicated to independent DApps. Therefore, the throughput of the entire network at 8 shards almost equals the sum of the throughputs of all shards.

Due to the dynamism of our sharding approach, the number of shards can be varied by creating or closing shards at runtime. Figure 4 depicts the throughput over time when new shards are created. Each shard consists of 8 machines and was evaluated in the Sydney AWS

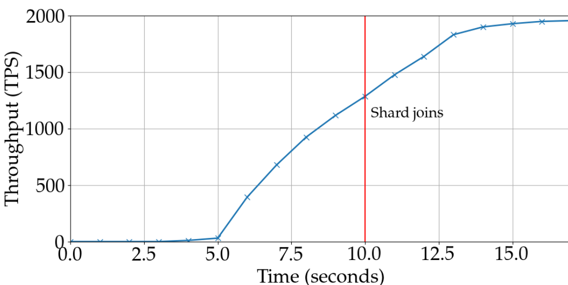

Figure 4 Throughput over time when new shards join.

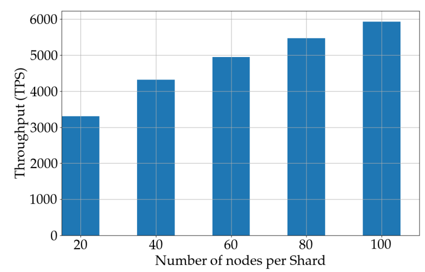

Figure 5 Throughput of 3 shards as their size increases.

region. At 0 second, there is only 1 shard. At around 10 seconds, another shard is created and starts processing transactions. Before the curve flattens and because a new shard starts processing transactions, the throughput keeps growing. Finally, the throughput stabilizes as expected when both shards keep processing transactions at full capacity. Note that the throughput of the mainchain is not considered as it only performs administrative tasks such as shard creation and shard rotation and does not process client transactions.

4.2 Dynamic node adjustment

With the dynamism we provide, the number of nodes in a shard can also be adjusted at runtime after an epoch time period, even when the total number of nodes in the network remains constant. Figure 5 illustrates this capability: We keep the number of shards fixed to 3 and vary the amount of nodes per shard. As can be seen, when the number of nodes per shard increases from 20 to 100, the total throughput also increases. At 50 participants per shard, a throughput of ∼6000 TPS is achieved. As CollaChain is known to be scalable [30] in that its performance grows with the provided resources, so does the throughput with the increasing number of nodes here. This makes our sharding approach particularly suited to run on top of CollaChain, so as to achieve dynamism while maintaining performance. Note that, the number of nodes per shard could have been increased further while maintaining performance due to CollaChain scalability.

5 Related Work

In this section, we present works related to blockchain sharding and previously summarized in Table 1. Section 5.1 lists the sharding protocols of blockchains offering native transfers of assets while Section 5.2 lists the sharding protocols of blockchains supporting smart contract, and thus DApp, execution. Interestingly, even the protocols for blockchains that support smart contracts do not invoke smart contract functions to reconfigure their shards. Some interesting works already create shards based on attributes (like locations [2]) while others rotate shards with randomization [8] like we do, however, we focus below on dynamism.

5.1 Payment Blockchains

Elastico [22] is the first sharded permissionless blockchain that tolerates byzantine failures. Elastico assumes synchrony and that at most 1/4 of the computational power is owned by byzantine participants. It mitigates Sybil attacks and shard take-overs with proof-of-work (PoW) and randomness, respectively, and rotates committees to tolerate static and

round-adaptive adversaries launching bribery attacks. Elastico upper bounds the number of validators per committee to 100, indicating a partial shard size dynamism, but it requires the number k of committees to be adjusted offline, which limits shard number dynamism. We are not aware of any mechanism to audit Elastico's current sharding configuration, like the number of validators.

OmniLedger [20] improves upon Elastico's decentralization and high failure probability and offers higher performance. Like for Elastico, Omniledger assumes synchrony, offers Sybil resistance via randomness and does not allow to audit the sharding configuration or to change the shard number at runtime. Omniledger rotates validators in each epoch using cryptographic sortition and a verifiable random function to mitigate bribery attacks. In addition of shards running their own instance of consensus, Omniledger also shards the blockchain state. Unlike Elastico, OmniLedger does not limit the number of validators, hence offering a higher degree of dynamism, yet it does not communicate transparently the number of validators to its users.

RapidChain [33] is the first sharded blockchain to support up to $f < n/3$ byzantine failures where n is the number of participants. Like Omniledger, RapidChain assumes synchrony and lets each shard maintain a portion of the blockchain state and run its own consensus instance. Candidate nodes solve a proof-of-work puzzle and create identities that they send to a reference committee, which randomly defines the next epoch committees. RapidChain allows nodes to join and leave the network and assigns them to existing shards, hence it offers a static shard number but a dynamic shard size.

SSChain [6] avoids the rotation of shard committees to shard the state without having to download the blocks and state. These data migrations, needed to verify transactions, can severly impact availability of the sharded blockchain. SSChain changes shards by allowing nodes to freely join, however, the risk is for a byzantine coalition to take over a shard. SSChain mitigates this attack by introducing a two-chained approach: a root chain verifies the blocks coming from each shard before committing them, which provides safety despite shard take-over at the condition of maintaining the entire state. SSChain offers neither transparency nor shard number dynamism but offers shard size dynamism.

5.2 DApp supported blockchains

Ethereum 2.0 (Eth2) is expected to introduce sharding to improve Ethereum's performance. Eth2 contains a fixed set of 64 shard chain and a single beacon chain [15]. Our approach is similar to Eth2 since we also request validators to escrow a deposit on the mainchain before assigning them to shard chains. However, Eth2 requires a minimum of 111 validators [31] to lower the probability of 2/3 adversarial nodes in a shard to 2^{-40}. At the end of each epoch, validators are rotated to maintain availability despite a slowly-adaptive adversary. Each shard runs a series of 64 Casper FFG consensus instances per epoch, after which a new block containing the shard states is appended to the beacon chain. Our approach differs from Eth2 by not forking, thanks to DBFT [7], not assuming synchrony, executing smart contracts even in the mainchain, and offering transparency and dynamism.

ChainSpace [1] is a transparent sharded blockchain that does not assume synchrony. An admin contract maps other smart contracts or "objects" to nodes that function as a shard, hence allowing users to consult the sharding configuration without inconsistencies. ChainSpace requires the admin contract creator to be trustworthy because if a shard contains a too large byzantine coalition, then the state of the blockchain could be compromised. ChainSpace offers transparency of the sharding configuration to its users and allows to dynamically adjust the number of nodes per shard, but cannot change the number of shards at runtime.

Zilliqa [29] exploits PoW and a random beacon to maintain two committees: one committee, called the "DS committee", is elected with PoW to create shards. Two pseudo-random numbers are generated: $r1$ comes from the last block in the previous DS committee while $r2$ comes from the last transaction block in a shard. The nodes solve an Ethash PoW cryptopuzzle based on their private key P_k, IP, $r1$ and $r2$. The first to solve this puzzle proposes a block that the DS committee agrees upon. Consequently, the successful miner is added to the DS committee and the oldest miner is churned out. At all times the DS committee has the most recent n miners. Zilliqa does not shard the state, assumes network synchrony and does not provide sharding dynamism or transparency.

Avalanche [27] offers "subnets" that can be viewed as shards. Three default subnets run three separate blockchains, the P-Chain handles metadata, the C-Chain handles native payments and the X-Chain handles smart contract executions. Avalanche offers dynamism because new subnet can be created and new validators can be added to an existing subnet. Unfortunately, Avalanche cannot work in a partially synchronous setting because its participants have to wait for the response of a small sample of nodes to progress, which could all be faulty [28]. Avalanche offers a JSON API to retrieve information about subnets, however, we are not aware of any verifiable way to collect tamper-proof information.

Cosmos [21] is a network of "zones" that can be viewed as shards as well. Each zone is a separate blockchain and the main one, called Hub, manages the governance of this network. Each zone builds upon the Tendermint consensus protocol that assumes partial synchrony and requires $f < n/3$ to solve consensus. Zones are not fully dynamic in that there cannot be more than a maximum of validators per zone, seemingly because performance decreases with the Tendermint participants. Even though the maximum number of validators per zone is announced to grow from 100 to 300 over a period of 10 years, one cannot add validators beyond this point. Although Cosmos offers information about validators [10] and zones [25] we are not aware of any way to guarantee this information is correct, as this information is not stored in the cryptographically secure ledger.

6 Conclusion

In this paper, we introduced dynamic blockchain sharding, the ability for a blockchain to change its sharding configuration at runtime without hard forks. Our implementation relies on smart contracts, hence anyone can double check the effectiveness of the reconfiguration by auditing the current state of the blockchain. The performance of our world-wide geo-distributed setting demonstrates that dynamic sharding scales quasi-linearly and can offer close to 14,000 TPS with only 8 shards.

References

1 Mustafa Al-Bassam, Alberto Sonnino, Shehar Bano, Dave Hrycyszyn, and George Danezis. Chainspace: A sharded smart contracts platform. *arXiv preprint*, 2017. arXiv:1708.03778.

2 Mohammad Javad Amiri, Divyakant Agrawal, and Amr El Abbadi. *SharPer: Sharding Permissioned Blockchains Over Network Clusters*, pages 76–88. Association for Computing Machinery, New York, NY, USA, 2021. doi:10.1145/3448016.3452807.

3 Nathalie Bertrand, Vincent Gramoli, Igor Konnov, Marijana Lazić, Pierre Tholoniat, and Josef Widder. Brief announcement: Holistic verification of blockchain consensus. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, July 2022.

4 Russell Brandom. $1.7 million in nfts stolen in apparent phishing attack on opensea users. Accessed: 2022-03-11. URL: https://www.theverge.com/2022/2/20/22943228/opensea-phishing-hack-smart-contract-bug-stolen-nft.

5 Jeff Burdges, Alfonso Cevallos, Peter Czaban, Rob Habermeier, Syed Hosseini, Fabio Lama, Handan Kilinc Alper, Ximin Luo, Fatemeh Shirazi, Alistair Stewart, and Gavin Wood. Overview of polkadot and its design considerations, 2020. arXiv:2005.13456.

6 Huan Chen and Yijie Wang. Sschain: A full sharding protocol for public blockchain without data migration overhead. *Pervasive and Mobile Computing*, 59:101055, 2019. doi:10.1016/j.pmcj.2019.101055.

7 Tyler Crain, Vincent Gramoli, Mikel Larrea, and Michel Raynal. DBFT: efficient leaderless Byzantine consensus and its application to blockchains. In *Proc. 17th IEEE Int. Symp. Netw. Comp. and Appl (NCA)*, pages 1–8, 2018.

8 Hung Dang, Tien Tuan Anh Dinh, Dumitrel Loghin, Ee-Chien Chang, Qian Lin, and Beng Chin Ooi. Towards scaling blockchain systems via sharding. In *Proceedings of the 2019 International Conference on Management of Data*, SIGMOD '19, pages 123–140, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3299869.3319889.

9 Luciano Freitas de Souza, Sara Tucci-Piergiovanni, Renaud Sirdey, Oana Stan, Nicolas Quero, and Petr Kuznetsov. Randsolomon: optimally resilient multi-party random number generation protocol. *arXiv preprint*, 2021. arXiv:2109.04911.

10 Big Dipper. Active validators. Accessed: 2022-03-15. URL: https://cosmos.bigdipper.live/validators.

11 C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):pp.288–323, 1988.

12 P. Ekparinya, V. Gramoli, and G. Jourjon. Impact of man-in-the-middle attacks on ethereum. In *Proc. 37th IEEE Int. Symp. Reliable Distrib. Syst. (SRDS)*, pages 11–20, October 2018.

13 Parinya Ekparinya, Vincent Gramoli, and Guillaume Jourjon. The Attack of the Clones against Proof-of-Authority. In *Proceedings of the Network and Distributed Systems Security Symposium (NDSS'20)*, February 2020.

14 The eth2 upgrades. Accessed: 2022-03-26. URL: https://ethereum.org/en/eth2/.

15 Ethereum. Shard chains. Accessed: 2022-03-15. URL: https://ethereum.org/en/upgrades/shard-chains/.

16 E. Fynn, A. Bessani, and F. Pedone. Smart contracts on the move. In *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 233–244, 2020. doi:10.1109/DSN48063.2020.00040.

17 Adem Efe Gencer, Robbert van Renesse, and Emin Gün Sirer. Short paper: Service-oriented sharding for blockchains. In Aggelos Kiayias, editor, *Financial Cryptography and Data Security*, pages 393–401, 2017.

18 Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, pages 51–68, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3132747.3132757.

19 Pedro Herrera. Dapp industry report – january 2022. Accessed: 2022-03-10. URL: https://dappradar.com/blog/dapp-industry-report-january-2022.

20 Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. Omniledger: A secure, scale-out, decentralized ledger via sharding. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 583–598, 2018. doi:10.1109/SP.2018.000-5.

21 Jae Kwon and Ethan Buchman. Cosmos white paper. Accessed: 2021-25-03. URL: https://v1.cosmos.network/resources/whitepaper.

22 Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. A secure sharding protocol for open blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 17–30, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2976749.2978389.

23 Satoshi Nakamoto. Bitcoin: a peer-to-peer electronic cash system, 2008.

24 C. Natoli and V. Gramoli. The balance attack or why forkable blockchains are ill-suited for consortium. In *47th IEEE/IFIP Int. Conf. Dependable Syst. and Netw. (DSN)*, June 2017.

25 Cosmos Networks. Map of zones. Accessed: 2022-03-10. URL: https://mapofzones.com/?testnet=false&period=24&tableOrderBy=ibcVolume&tableOrderSort=desc.

26 randao.org. Randao: Verifiable random number generation. Technical report, randao.org, 2017. Accessed February 2022. URL: https://www.randao.org/whitepaper/Randao_v0.85_en.pdf.

27 Team Rocket. Snowflake to avalanche: A novel metastable consensus protocol family for cryptocurrencies. Technical report, Avalanche Foundation, 2018. Accessed: 2021-12-01. URL: https://ipfs.io/ipfs/QmUy4jh5mGNZvLkjies1RWM4YuvJh5o2FYopNPVYwrRVGV.

28 Team Rocket, Maofan Yin, Kevin Sekniqi, Robbert van Renesse, and Emin Gün Sirer. Scalable and probabilistic leaderless bft consensus through metastability. Technical Report 1906.08936v2, arXiv, 2019. arXiv:1906.08936v2.

29 The ZILLIQA Team. The zilliqa technical whitepaper. Technical report, Zilliqa, 2017. Accessed February 2022. URL: https://docs.zilliqa.com/whitepaper.pdf.

30 Deepal Tennakoon, Yiding Hua, and Vincent Gramoli. Collachain: A bft collaborative middleware for decentralized applications, 2022. arXiv:2203.12323.

31 SJ Wels. Guaranteed-tx: The exploration of a guaranteed cross-shard transaction execution protocol for ethereum 2.0. Master's thesis, University of Twente, 2019.

32 Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger, 2015. Yellow paper.

33 Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. Rapidchain: Scaling blockchain via full sharding. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, pages 931–948, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3243734.3243853.

Analyzing Soft and Hard Partitions of Global-Scale Blockchain Systems

Kevin Bruhwiler ✉
University of California, Irvine, CA, USA

Fayzah Alshammari ✉
University of California, Irvine, CA, USA

Farzad Habibi ✉
University of California, Irvine, CA, USA

Juncheng Fang ✉
University of California, Irvine, CA, USA

Faisal Nawab ✉
University of California, Irvine, CA, USA

Abstract

Partitioning attacks have been a known threat since the invention of cryptocurrencies. Attackers could deliberately fork the chain by re-routing network traffic into two or more separate chains and spend money on each piece, effectively spending multiples of their money. Apostolaki et. al. [1] were among the first to quantify the threats of such attacks on Bitcoin. They suggest a number of ways to mitigate this risk which were combined into a tool named SABRE.

Jyothi explored the possibility that a solar superstorm could damage the undersea fiber-optic cables that connect the Internets of different continents, and considered the mostly likely ramifications of the damage. She concluded that such an event would likely cause major connectivity issues across the northern hemisphere and may disconnect much of North America's internet from the eastern hemisphere for weeks. There is also concern that undersea cables could be deliberately destroyed as acts of terrorism or war or by natural disasters such as earthquakes.

In this work, we construct a simulation to properly quantify the effects of a global-scale network partition on the blockchain. We hope to provide the groundwork for preventative measures to be taken to minimize the harm that such partitions might cause in the future. We do this by modifying SimBlock [2], a blockchain simulator created to study the effect of different network topologies, to allow initiating and recovering from partitions and also add metrics to capture their effects.

To quantify the severity of partitions we use a number of metrics, including the rate of agreement improvement after a new block has been minted and the average rate of block propagation across regions. We also examine the number of forks in the blockchain that result from partitions and identify the break-points at which forks begin to appear. Finally, we quantify the duration that partitions of various sizes can persist before they begin to generate forks and measure the how long it takes for the system to recover once the partition has been resolved.

2012 ACM Subject Classification Computer systems organization → Reliability; Computer systems organization → Peer-to-peer architectures

Keywords and phrases Blockchain, Partitioning, Resilience, Simulation

Digital Object Identifier 10.4230/OASIcs.FAB.2022.7

Category Poster

References

1. Apostolaki, Maria Aviv, Zohar and Laurent Vanbever. Hijacking bitcoin: Routing attacks on cryptocurrencies. *IEEE Symposium on Security and Privacy (SP). IEEE*, 2017.
2. Yusuke, Aoki et al. Simblock: A blockchain network simulator. *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS). IEEE*, 2019.

© Kevin Bruhwiler, Fayzah Alshammari, Farzad Habibi, Juncheng Fang, and Faisal Nawab;
licensed under Creative Commons License CC-BY 4.0

A Modular Approach for the Analysis of Blockchain Consensus Protocol Under Churn

Floris Ciprian Dinu ✉
Department of Computer, Control, and Management Engineering Antonio Ruberti,
University of Rome La Sapienza, Italy

Silvia Bonomi ✉
Department of Computer, Control, and Management Engineering Antonio Ruberti,
University of Rome La Sapienza, Italy

Abstract

Blockchain is an emerging technology that gained a lot of attention in the last years. Many different consensus protocols have been proposed to improve both the scalability and the resilience of existing blockchain. However, all these solutions have been defined for rather static settings. We propose a modular approach for analysing and comparing different consensus protocols used in blockchain under churn.

2012 ACM Subject Classification Computer systems organization → Reliability; Information systems → Distributed storage

Keywords and phrases Blockchain Dependability, Dynamic Distributed Systems, Simulation

Digital Object Identifier 10.4230/OASIcs.FAB.2022.8

Category Poster

Introduction. In the last 10 years, Blockchain became one of the most widespread technology used to store transactions in a distributed system characterized by full decentralization, transparency, immutability and non-repudiation of data. Blockchain represents an example of emerging technology that first consolidated its development and only recently started to investigate the theoretical foundations behind them. As a consequence, many different algorithmic solutions have been defined trying to improve as much as possible the scalability and the resilience to Byzantine processes. However, most of the existing solutions lack a solid theoretical analysis proving their formal correctness and the evaluation is carried out by considering rather static environments where the system does not change or changes very slowly mainly due to failures. However, real networks (especially those underlining public permissionless blockchain) are not static and are subject to a progressive refreshment of the peers participating in the system. Such phenomenon is also known as *churn* and if not properly analysed and managed may have a strong impact on both correctness and performance of the blockchain. To the best of our knowledge, currently there do not exist results showing the impact of churn over the blockchain. We took a first step in this direction by defining a framework that can be used to evaluate how existing consensus protocols for blockchains respond to churn.

Reviewing and analysing the state of the art on consensus protocols for blockchain [6], we observed that every blockchain protocol can be seen as the composition and orchestration of the following main distributed building blocks:

- an *Overlay Management Protocol* (OMP) responsible for connecting replicas into a logical overlay network and preserve the connectivity of the overlay network graph;
- a *Communication Layer* implementing one-to-one, one-to-many and many-to-many communication primitives that allow the dissemination of transactions and blocks to all interested replicas and

© Floris Ciprian Dinu and Silvia Bonomi;
licensed under Creative Commons License CC-BY 4.0
5th International Symposium on Foundations and Applications of Blockchain 2022 (FAB 2022).
Editors: Sara Tucci-Piergiovanni and Natacha Crooks; Article No. 8; pp. 8:1–8:2
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

- an *Agreement* primitive (e.g., a consensus, a leader election, a committee-based voting) that is used to select, validate and attach blocks to the blockchain consistently with other replicas in the system.

Let us note that such primitives are not independent of each other but they rather work in synergy. As a consequence, when the system becomes dynamic the effect of the churn does not impact only the overlay network and the OMP but it also impacts all the other layers built on top of it.

Research Direction and Contribution. Our research is aimed to define a framework that can be used to analyse different blockchain solutions in dynamic settings and to compare their characteristics. Our proposed framework is composed of four main elements:

- a *distributed building blocks composition model* that allows to define a blockchain protocol as composition of existing distributed building blocks.
- a *churn* model that allows to characterize the dynamic of the system and to describe the arrival and departure distribution of processes from the system (and in particular at the OMP level).
- a *load* model that allows to characterize the how transactions are generated by clients
- a set of *metrics* that allows to analyse every blockchain protocol and to perform a comparison between different protocols.

To create our composition model, we selected consensus protocols from the state of the art (e.g., SCP [4], Tendermint [1], XRP [3], PBFT [2]) and we analysed them to identify the set of assumptions concerning (i) the overlay network (ii) the communication primitives used and (iii) the type of agreement implemented on top of them. Almost every algorithm either assumes a fully connected overlay network (e.g., PBFT, Tendermint) or an overlay network having the characteristics of a random graph (e.g., SCP). Concerning the communication primitives, almost all the papers consider a reliable communication system without specifying any further detail about its implementation. Then we analysed the state of the art concerning OMPs and protocols implementing a reliable communication primitive. While from the correctness point of view the many available solutions can be considered equivalent each other, from the performance, dependability and robustness point of view they are not. Thus, we implemented in OMNeT++ [5] a composition model that allows to define a blockchain as a composition of (i) one OMP, (ii) one (or more) communication primitive(s) and (iii) an agreement primitive and we are currently implementing several algorithms, exposing the same interface, for any required building block.

References

1 Ethan Buchman, Jae Kwon, and Zarko Milosevic. The latest gossip on BFT consensus. *CoRR*, abs/1807.04938, 2018. arXiv:1807.04938.

2 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, USA, 1999. USENIX Association.

3 Brad Chase and Ethan MacBrough. Analysis of the XRP ledger consensus protocol. *CoRR*, abs/1802.07242, 2018. arXiv:1802.07242.

4 David Mazieres. The stellar consensus protocol: A federated model for internet-level consensus. *Stellar Development Foundation*, 2015.

5 OMNeT++. Home page, 2022. URL: https://omnetpp.org.

6 Yang Xiao, Ning Zhang, Wenjing Lou, and Y Thomas Hou. A survey of distributed consensus protocols for blockchain networks. *IEEE Communications Surveys & Tutorials*, 22(2):1432–1465, 2020.

Improving Blockchain Resilience to Network Partitioning by Sharding

Juncheng Fang ✉
University of California, Irvine, CA, United States

Farzad Habibi ✉
University of California, Irvine, CA, United States

Kevin Bruhwiler ✉
University of California, Irvine, CA, United States

Fayzah Alshammari ✉
University of California, Irvine, CA, United States

Faisal Nawab ✉
University of California, Irvine, CA, United States

Abstract

Blockchain plays a significant role in cryptocurrencies and growing applications like smart contracts. However, prior blockchain algorithms did not consider large-scale network partitioning a considerable concern while relying heavily on a reliable global network. Previous works have shown a possibility of a massive disruption on the Internet. The author in [2] discusses the case of Internet disorder due to solar superstorms, which can disconnect different geographical regions from each other for months. Partitioning attacks are also notable concerns that should be considered, in which their goal is to cut connections between a set of nodes and the rest of the network.

In the case of network partitioning, the main chain will fork into branches, and miners in different disconnected regions will create multiple blocks in parallel. The longest chain rule in current blockchain systems accepts only one of the branches after the network is recovered, and because of that, all blocks in other branches will be pruned. Losing a considerable number of mined blocks is not tolerable and significantly impacts the reliability of the ledger and miners' benefit.

In this work, we aim to improve blockchain resilience by designing a partition-tolerance blockchain system that: (1) split into branches when network partition happens. (2) merge existing branches into one when the network goes back to normal. (3) ensure the safety and integrity of the blockchain.

Newly mined blocks will be collectively signed by a group of miners with a BFT protocol similar to ByzCoin[1], where the consensus group is formed by the miners of the previous w blocks. When a network partition happens, only part of the consensus group can be reached; thus the number of signers w_b of the new block will be less than w. If a block with w_b signers is published, every node in the partition learns that they are now in a branch with around w_b/w of the total hashing power, and it can be identified by the signature of the block. After the network recovers, miners will receive multiple branches, and they mine on a merging block which points to the last block of each branch as the parent blocks. The consensus group will be selected from each branch according to the branch size. Transactions in each partition are preserved after merging.

2012 ACM Subject Classification Computer systems organization → Reliability

Keywords and phrases resilience, partitioning, blockchain, collective signing

Digital Object Identifier 10.4230/OASIcs.FAB.2022.9

Category Poster

References

1. Kogias et al. Enhancing bitcoin security and performance with strong consistency via collective signing. In *25th usenix security symposium (usenix security 16)*, pages 279–296, 2016.
2. Sangeetha Abdu Jyothi. Solar superstorms: planning for an internet apocalypse. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, pages 692–704, 2021.

© Juncheng Fang, Farzad Habibi, Kevin Bruhwiler, Fayzah Alshammari, and Faisal Nawab; licensed under Creative Commons License CC-BY 4.0

5th International Symposium on Foundations and Applications of Blockchain 2022 (FAB 2022).
Editors: Sara Tucci-Piergiovanni and Natacha Crooks; Article No. 9; pp. 9:1–9:1
OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Why General Collective Intelligence Must Be the Future of the Blockchain

Andy E. Williams
Nobeah Foundation, Nairobi, Kenya

Abstract

General Collective Intelligence or GCI is predicted to radically increase the speed and scale at which blockchain technology can be designed, developed, and deployed as well as being predicted to radically increase demand for those new blockchain based products and services where they don't involve consumption of limited physical resources. Therefore, if a GCI can be implemented, it is predicted that GCI based platforms will quickly come to dominate the blockchain marketplace and that GCI is the future of the blockchain. But it also must be the case that GCI is the future of the blockchain because without it, through an effect called the "technology gravity well" blockchain and other technologies have the possibility of introducing an unprecedented degree of centralization, control, and abuse.

2012 ACM Subject Classification Human-centered computing → Human computer interaction (HCI)

Keywords and phrases General Collective Intelligence, Human-Centric Functional Modeling, functional state space, conceptual space, blockchain state space, cooperation state space

Digital Object Identifier 10.4230/OASIcs.FAB.2022.10

Category Poster

Related Version SoK: Is General Collective Intelligence the Future of the Blockchain?
Full Version: https://osf.io/preprints/africarxiv/u7jaz/

1 Functional State Space

Unlike ontologies, state spaces have the potential to represent any possible states of a system and any possible process (behavior) that might be used to transition between those states, in the same way that three dimensional Euclidean space has the potential to represent any possible positions of a system and the motions that might be used to transition between those positions. Functional state space differ from other state spaces in that a functional state space is spanned by some minimal set of functions so that all functional states can be expressed as some composition of that minimal set [3]. Any ontology describing the behavior of a system can potentially be viewed as a subset of some functional state space, where that subset is chosen by some centralized actor. Ontologies unlike functional state spaces are then inherent sources of centralization.

2 Functional State Space and Exponentially Increasing General Problem-Solving Ability

All systems can have their behaviors (functions and processes) described in terms of functional state spaces. For any system described in terms of a functional state space, problems in understanding the system are represented as the lack of a process capable of transitioning the system from one state to another state. General problem-solving ability is represented as the volume of functional state space that can be navigated per unit time, multiplied by the density of functional states that must be navigated. Any system, including blockchain

© Andy E. Williams;
licensed under Creative Commons License CC-BY 4.0
5th International Symposium on Foundations and Applications of Blockchain 2022 (FAB 2022).
Editors: Sara Tucci-Piergiovanni and Natacha Crooks; Article No. 10; pp. 10:1–10:3
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

platforms, whose behaviors are expressed in terms of functional state spaces, can achieve an exponential increase in ability to solve problems through the same pattern of solution in functional state space [4]. This pattern of solution is General Collective Intelligence [2].

3 General Collective Intelligence (GCI)

A General Collective Intelligence or GCI is a hypothetical platform able to orchestrate groups of individuals or intelligent agents in collectively executing any possible reasoning, and able to organize any possible cooperation that might increase the complexity of the collective reasoning groups are able to execute, where that cooperation might also increase the speed and scale at which that reasoning might be executed. GCI defines a universal representation of concepts and reasoning that can be shared between all participants in collective reasoning. This "functional state space" of the collective cognition (called the "collective conceptual space") is hypothesized to have the ability to represent all possible reasoning processes, and therefore all possible behaviors of this collective cognition. GCI also defines a cooperation state space capable of representing all possible cooperation processes through which the outcomes of that reasoning might be scaled. A GCI then has the potential to orchestrate the execution of any possible collective reasoning and to do so through any possible process of cooperation.

4 Applying Functional State Space and GCI to the Blockchain

All software can potentially be represented in terms of a hierarchy of functional state spaces, including identity management, data management, the blockchain functional state space, and other functional state spaces. If so then blockchain platforms can be represented by a set of paths through blockchain state space. By decoupling blockchain platforms into a library of functional components that are each represented by a path segment through blockchain functional state space, then a GCI might exponentially increase ability to solve any problem in that domain through assembling those path segments in order to navigate from any initial functional state to any target functional state. Since a GCI can potentially orchestrate cooperation between what might be billions of individuals, or even billions of intelligent agents working on behalf of each individual, this self-assembly and adaptation of software might take place dynamically and at orders of magnitude greater speed and scale than humans could possibly achieve, radically increasing the speed and scale at which blockchain technology can be developed, along with radically increasing the capacity of developers to solve any blockchain problem, including increasing blockchain interoperability, or cryptocurrency deployment while simultaneously increasing speed, scale, and security [1].

5 The Technology Gravity Well

GCI is a general system of decentralized decision-making that can be applied to any process along the entire life-cycle of blockchain platforms from research and development to administration. Without it, some blockchain platform related processes (usually design and administration) tend to be centralized. Any technology that mediates interactions within a group is centralized where it constrains decision-making to be aligned with the interests of some subset of that group. Due to an effect called "the technology gravity well" [5] in the absence of a general system for decentralization like GCI there is predicted to be an irreversible free fall towards centralization with the advance of technology. In the case of

blockchain and other technologies for which decentralization is the main selling point, this centralization might be invisible because it's natural to assume centralization isn't there, because that centralization is too complex for most to see, and because this centralization happens faster than it can reliably be detected and removed. This technology gravity well has dire societal implications, namely centralization to the point that there can be no possibility of social protection against even the worst transgressions. GCI is the only known mechanism through which it is predicted to be possible to escape this technology gravity well.

6 Larger Societal Importance

In this paper it has been hypothesized that all systems and all properties of systems can be understood in terms of functional state spaces. If so then defining a blockchain functional state space to represent all possible functions of platforms within the blockchain domain might make it possible to define expressions for properties like complexity that apply to whatever objects represent the functional states of that blockchain domain. In the same way, defining a functional state space (the conceptual space) to represent all possible functions of the cognitive system might make it possible to define expressions for properties like "importance" that apply to concepts as the functional states of that domain. Speaking about the property of importance specifically, it is hypothesized that the importance of a tool can potentially be understood in terms of the volume of conceptual space it allows to be navigated. If so, then in exponentially increasing the navigable volume of conceptual space, where simple geometrical arguments in conceptual space suggest this exponential increase has never been possible before, GCI might be the most important innovation in the history of human civilization and one that might radically increase our capacity to solve every collective challenge from poverty to climate change [1]. However, due to a great many factors coming together, no one yet knows about GCI and fewer still understand it. It is hoped this short introduction might encourage more researchers to take up the challenge of validating that GCI can be applied to the blockchain, and to generalize their example so it can be used to validate whether GCI can be applied to solve other problems in other functional state spaces like the conceptual space, such as accelerating progress towards the sustainable development goals.

References

1. Andy E Williams. Cognitive computing and its relationship to computing methods and advanced computing from a human-centric functional modeling perspective. In *SCRS Conference Proceedings on Intelligent Systems, SCRS, New Delhi, India*, pages 16–33, 2021.
2. Andy E Williams. Defining a continuum from individual, to swarm, to collective intelligence, and to general collective intelligence. *International Journal of Collaborative Intelligence*, 2(3):205–209, 2021.
3. Andy E Williams. Human-centric functional modeling and the unification of systems thinking approaches: A short communication. *Journal of Systems Thinking*, 2021.
4. Andy E Williams. Automating the process of generalization, March 2022. doi:10.31730/osf.io/fb4us.
5. Andy E Williams. Breaking through the barriers between centralized collective intelligence and decentralized general collective intelligence to achieve transformative social impact. *International Journal of Society Systems Science*, 2022.