



Dynamic membership management for a permissioned blockchain

Hafsteinn Hjartarson¹ · Fjölínir Thrastarson¹ · Anna Sigridur Islind¹ · Gísli Hjálmtýsson¹

Received: 30 June 2024 / Revised: 24 February 2025 / Accepted: 2 March 2025 / Published online: 30 August 2025
© The Author(s) 2025

Abstract

Permissioned blockchains have gained prominence as a means of decentralizing trust while retaining controlled access, particularly in enterprise settings and regulated peer-to-peer environments. These systems offer advantages in scalability, performance, and security; however, challenges persist in effectively managing membership and its interaction with consensus protocols. Ethereum's transition to Proof-of-Stake has also been a transition to managed membership, where validators are actively monitored and penalized for non-performance. This paper examines the dynamic tension between membership management and consensus protocols in permissioned blockchains, as well as the benefits of active management in improving overall system performance. In this paper, we propose a framework for dynamic membership management that includes actively admitting, monitoring, and ejecting members. Our approach decouples membership management from the underlying blockchain construction process. Our simulations confirm the potential benefits of managed membership, in part to facilitate lightweight mechanisms for improved performance and reliability. Our findings suggest that dynamic membership management is a critical area of study with significant implications for the future design of permissioned blockchains. Our contributions provide a conceptual foundation for designing dynamic membership protocols in permissioned blockchains, filling a gap in the literature and offering practical solutions to enhance blockchain performance in controlled environments.

Keywords BFT · Blockchain · Membership management

1 Introduction

Permissioned blockchains have gained traction lately as a way to decentralize and distribute trust while maintaining a certain level of control over the access of the underlying blockchain [1]. In many cases, permissioned blockchains are intra-enterprise systems, with all members of the

blockchain belonging to or being managed by the enterprise. Unlike permissioned systems, where anyone can join the network, the construction of the chain and its data, permissioned blockchains may restrict access to certain levels of the system's functionality and features. Within this realm, there are successful cases such as Ripple's XRP Ledger [2], Corda [3], and Hyperledger Fabric [4]. Even if owned and controlled by a forming entity, such permissioned blockchains may be open to anyone satisfying certain criteria and will, in general, attract more participants and thus increase in value by adopting open, transparent, and decentralized operating rules. The membership itself will be the primary benefit for those blockchains embodying valuable services to their members.

An important challenge in permissioned blockchain systems is managing the participating members. This management entails how new members are added and removed to and from the network, as well as their allowed interactions with the system as a whole. Robust membership management is, therefore, an integral part of maintaining system security and compliance and can aid in optimizing the underlying consensus mechanism. To

Hafsteinn Hjartarson, Fjölínir Thrastarson, Anna Sigridur Islind, and Gísli Hjálmtýsson have contributed equally to this work.

✉ Hafsteinn Hjartarson
hafsteinn24@ru.is

✉ Gísli Hjálmtýsson
gisli@ru.is

Fjölínir Thrastarson
fjolnir17@ru.is

Anna Sigridur Islind
islind@ru.is

¹ Department of Computer Science, Reykjavík University, Menntavegur 1, Reykjavík 102, Iceland

underline this, Ethereum's [5] shift from Proof-of-Work (POW) to Proof-of-Stake (POS) is a shift towards managed membership where each participating node is now known and may be penalized or excluded from the network based on its performance. As already evident in POS Ethereum, the uncontrolled membership size gives rise to several scaling and performance issues, many of which disappear or are significantly reduced by restricting membership. Further, some permissioned systems, such as Ethereum and Corda [3], make no assumptions about the reliability or failures on the set of permissioned nodes, once admitted, and employ a full byzantine fault tolerant consensus protocol with all the complexity and messaging overhead that it entails [6, 7]. Currently, different permissioned systems rely on ad-hoc mechanisms to control and manage membership without a formalized membership governance.

In [8–10], the authors conclude that using a Membership Authority (MA) to manage the permissioned set may facilitate the adoption of more lightweight consensus protocols compared to their permissionless counterpart with unrestricted writer set. In this paper, we describe a membership authority framework that admits, monitors and removes members based on transparent policies, operating at timescales one or two orders of magnitude larger than the block construction. These members become writers, collaborating in the blockchain's construction. A comprehensive membership protocol must address the inclusion and exclusion of writers, engage with writer compliance and performance monitoring, and play an integral role in both rewarding and penalizing mechanisms. A key design issue is understanding how this monitoring and management influence the consensus protocol and overall system performance. This paper aims to fill a gap in the literature by exploring these design challenges, the interaction between the two layers, and the level of synchrony needed between the layers. Our contributions resonate well with recent developments in several blockchains, as a major contribution herein is the identification of mechanisms, policies, and their interactions supported by our lightweight consensus protocol.

We propose our membership management framework using our lightweight blockchain. However, we do not see that as a specific case; instead, we use it to extrapolate a more generalized knowledge of dynamic membership management. We base our findings on the notion that although the knowledge to date on blockchains, in general, is vast, the knowledge of what dynamic membership management contains is missing in the literature to date. We pose the research question: *What constitutes dynamic membership management, and how can dynamic membership management for permissioned blockchains be designed?* Against this backdrop, we derive a two-fold contribution. Firstly, we contribute to the literature with the

conceptualization of dynamic membership management. Secondly, we provide the framework outlined herein, which others can use when designing dynamic membership management for a permissioned blockchain. In Sect. 7, we show that the rate of failures is negligible and further analyze the performance of our design.

The rest of the paper is organized as follows. We start with the background in Sect. 2, establishing the nomenclature used in this paper before a related work overview and contrasting in Sect. 3. We then give the design of our membership management framework in a permissioned blockchain in Sect. 4, followed by a summary description of our experimental blockchain Annáll in Sect. 5. We then dive into the online management of the penalty box in Sect. 6, which is a crucial part and contribution, as it couples the mechanisms of the membership maintenance on the one hand and the construction of the blockchain and the consensus protocol on the other. Analysis and simulations are described in Sect. 7. The results and observations are discussed in Sect. 8, followed by our conclusions in Sect. 9.

2 Background

2.1 The blockchain system

Blockchains can generally be categorized into three types: (i) permissioned, (ii) permissionless blockchains, and (iii) consortium blockchains. While permissionless blockchains allow open participation, enabling anyone to take on roles in the construction or use of the blockchain system, permissioned- and consortium blockchains can impose rules that regulate who may participate and in what capacity [11]. Because of the controlled nature of permissioned blockchains, they attract the attention of organizations looking to transition into disruptive decentralized solutions while offering a safe and reliable way to share data [12].

A blockchain system can be defined as a set of nodes working toward the common objective of writing, confirming, and verifying a single, totally ordered chain of transaction blocks, which is, by construction, immutable and verifiable by third parties. We consider a blockchain decentralized if all nodes have an equitable opportunity to contribute to the chain. No further assumptions are made regarding the blockchain's structure, the content of the blocks, or the underlying service model beyond the basic framework outlined above.

A subset of nodes, herein called writers, is responsible for constructing and adding new blocks to the chain. Another subset is involved in the process of selecting a winner. Whereas these two sets can theoretically be distinct, they are often identical, which is the assumption

made in this paper. Additionally, we distinguish two other subsets: clients and auditors, who have access to the entire blockchain and can verify its integrity. Auditors, in particular, strengthen the immutability of the chain by offering third-party verification without introducing additional complexity to the consensus process. In the seminal Bitcoin [13] paper, only three sets are defined: writers, winner selectors, and clients, with the assumption that all nodes perform all of these roles.

The authors note that the process of constructing the chain is self-clocked, allowing at most one block to be added to the chain per round. During each round, one node is selected as the *winner*,¹ who constructs and disseminates the new block.

2.2 Consensus protocols

At the heart of every blockchain system is its consensus mechanism, selecting the winner to write the next block. A number of consensus mechanisms have been proposed for blockchain, most notably Proof-Of-Work [13], originally proposed for Bitcoin and initially used for Ethereum, and Proof-Of-Stake [5] as currently employed by Ethereum. In addition, there is significant prior work on leader-based distributed state replication and Byzantine Fault Tolerance (BFT) [14], including the Ripple Protocol Consensus Algorithm (RPCA) [15], Paxos [16], Raft [17], and Practical Byzantine Fault Tolerance (PBFT) [18]. However, blockchain differs from prior work on state replication in that a new leader is selected for every round, thus changing the engineering parameters. In contrast, the traditional state replication solutions [16–18] only elect a new leader when the current one fails.

Recognizing this, some blockchains have developed their own version of consensus [4, 15, 19]. We observe that the complexity of consensus to select the next leader is $O(n^2)$ in the number of nodes participating in the consensus protocol. Multiple other consensus protocols have been proposed [20, 21], specifically aimed at solving challenges such as Byzantine fault tolerance, access control, and interoperability [22–25]. Moreover, immutability is achieved through third-party verifiability and only requires an auditing function. Hence, adding nodes only doing auditing and thus policing the writers enhances the chain's value at minimal cost.

Table 1 shows the message complexity comparison between three popular blockchain systems (XRPL, Ethereum, and Tendermint) and our elementary blockchain system, Annáll, used for our prototyping, and described further in Sect. 5. All of these establish a partial-mesh

network among the peer-to-peer members. Whereas Ethereum introduced a form of dynamic membership management when shifting to POS, XRPL is currently statically maintained. Tendermint, by design, includes membership management. Annáll is listed as using Dynamic Membership Management (DMM) by virtue of the prototype implementation of the Membership Authority (MA) discussed herein. Whereas the message complexity of Annáll is $O(n)$, the other three grow with $O(n^2)$.

2.3 Membership management

For permissioned blockchains, an additional function is managing the set of participating nodes, which may change dynamically for several reasons. More nodes than are needed may desire to access the network, which, if unchecked, may ultimately be to the detriment of the system. Meanwhile, unpredictable and dynamic changes to membership are assumed to occur at timescales, several orders of magnitude larger than transactions.

In this paper, we propose a framework that encompasses membership entry and exit functionality, whether voluntary or forced, and a *penalty box* for monitoring, penalizing, and potentially removing misbehaving nodes from the active writer set. In our framework, persistently non-performing or non-compliant members are removed, which only temporarily impacts the behavior and performance of the system. As a consequence, the finer time scale functions, in particular consensus, can be optimized based on optimistic assumptions for the common case, where the protocol progresses correctly. Clearly, failures occur and must be appropriately addressed. However, as long as such failures are infrequent, our approach is to handle failures as true exception handling.

To research the framework, we have built an experimental blockchain, Annáll, exploiting that the member set is known to all participating nodes. In this blockchain and defined further in Sect. 5, each node maintains a set of active writers online. At the beginning of every round, a sufficient part of the active writers are in agreement on the active writer set, which participates in the round for consensus. Failing nodes, whatever the cause, enter the penalty box and are removed (temporarily or permanently) from the active writer set and excluded from the consensus. Using the Annáll blockchain, we have prototyped all aspects of the framework and used it with several blockchain services, including payments, a digital certification service, and a traceability service.

Nodes may be rewarded for writing to the block or otherwise desire to write to the block but may misbehave to gain an advantage or attempt to rewrite part of the chain. Clearly, nodes can fail and may maliciously attempt to

¹ The term “winner” is preferred over “leader” since the selected node only leads for one round.

Table 1 Comparison of selected permissioned blockchain consensus protocols

Blockchain	Network type	Membership management	Complexity
Annáll	Partial-mesh	Dynamic	$O(n)$
XRPL	Partial-mesh	Static	$O(n^2)$
Ethereum	Partial-mesh	Dynamic	$O(n^2)$
Tendermint	Partial-mesh	Dynamic	$O(n^2)$

game the system or simply prevent it from functioning. We observe that for those blockchains providing valuable service, access to the services of the blockchain is the primary value, and that value increases as n^2 where n is the number of value-creating members. In the case of a direct rewarding mechanism, we observe that the chain contains the identity of each block's creator (or its wallet) and thus need not be rewarded immediately or online. In some cases, the membership authority would seem the appropriate venue for such rewards.

3 Related work

Despite the significant body of literature on blockchain technology, the issue of dynamic membership management remains unsolved. In permissionless blockchains such as Bitcoin, membership is entirely open and unmonitored, and the concept of dynamic membership is not a central concern. However, much of the existing research on blockchain technology is derived from Bitcoin's architecture, leaving a gap in the literature on dynamic membership management in other blockchain applications. Prominent blockchains, including Ethereum, Tendermint/Cosmos, and the XRP Ledger (XRPL), employ distinct methods for the addition of new validators. Yet, no consensus or unified approach to membership management has emerged within the field. This fragmentation in membership management strategies underscores an important research opportunity for dynamic membership management addressing challenges such as trust, scalability, and cross-chain interoperability.

3.1 Ethereum

In Ethereum, all validators participate in selecting the next winner. The consensus rounds are split up into *slots* and *epochs*, where one slot is 12 seconds, and one epoch is 32 slots. In each slot, however, a proper subset of validators is selected as a committee participating in the winner selection process, randomly selecting a *proposer* and attesting to the new block proposal. The proposer constructs and proposes the next block, which, if successfully validated, gets

added to the chain. If any steps fail, the slot completes without constructing a new block.

Ethereum has recently gone through an interesting transformation. Initially, Ethereum shared similarities with the Bitcoin blockchain, as membership was completely open and unmanaged, using POW as consensus and differing from Bitcoin only in its enriched service model [26]. The move to POS is fundamental for resource use and equally transformational since the Ethereum blockchain now has managed membership. Although Ethereum is still open to anyone willing to pay the minimum stake, validators are not only known but are actively monitored and potentially penalized for misbehavior or non-performance through *slashing* [27].

The shift to POS was driven by the desire to reduce the computational complexity of the consensus protocol and its associated resource consumption. However, it has introduced significant complexity in both membership management and the POS consensus mechanism, as well as in the interaction between them. In this paper, we argue that in all blockchains with managed membership, there is an inherent tension between membership management and the underlying consensus protocol. However, managed membership may also introduce opportunities for simplification at the blockchain level by leveraging information from the management layer, which could enhance the overall efficiency of blockchain construction. From this paper's point of view, Ethereum represents an example of dynamic membership management. We argue that this topic warrants further formal studies exploring the relative merits of basic mechanisms and various policy choices when managing a set of writers on a chain.

With the adoption of POS as its consensus mechanism, Ethereum requires validators to stake a specified amount of ETH, the network's native cryptocurrency, to activate their validator software [5]. Anyone may join as a validator by providing the required stake independently or via a third-party service. Ethereum enforces strict conditions for validators, including maintaining honesty (e.g., not proposing conflicting blocks), ensuring uptime, and meeting hardware requirements. Failure to meet these standards results in *slashing*, a punitive measure that reduces a validator's staked ETH, either partially or fully. If a validator's stake

falls below the minimum threshold of 32 ETH, they are disqualified from the validator set.

Although Ethereum's *stake* and *slash* mechanism is a notable example of monitored membership that actively monitors and manages participants through staking, it presents potential risks to both the stability and decentralization of the system. Recent analyses suggest that large staking pool services are approaching control of one-third of the total stake, raising concerns about the centralization of power within the platform [28].

Despite the theoretical preference for solo staking, users are incentivized to join staking pools due to factors such as operational simplicity and reduced time to market. For instance, users who possess the required stake often opt for pooled staking to bypass Ethereum's validator entry queue and gain immediate access to rewards. This mechanism, while efficient in some respects, poses a significant challenge to decentralization, as it allows wealthier participants or large staking pools to unduly exert control over the network.

In contrast, membership authorities could introduce safeguards to preserve decentralization by imposing constraints on the distribution of power, including regulating the maximum number of participants in the writer set or implementing other controls based on the preferences of the chain's governance framework.

3.2 XRPL

In Ripple's consensus protocol, each server maintains a Unique Node List (UNL), which is a set of trusted validators that the server believes will not collude [2]. By default, a node setup employs two UNLs: one provided by the XRP Ledger Foundation and another by Ripple. Each server uses its UNL to determine which validators' consensus votes it will consider during the consensus process. Concurrently, the server monitors the reliability of the validators in its UNL, where reliability is measured as the percentage of the validator's last 256 votes that align with the server's view of the consensus.

To enhance the system's liveness, Ripple utilizes a mechanism known as the Negative Unique Node List (Negative UNL). If a validator's reliability falls below 50%, the server removes the validator from its UNL and places it in the Negative UNL. Consequently, the server no longer includes the validator's votes in the consensus process but continues to track the validator's reliability score. If the validator's reliability exceeds 80%, the validator is reinstated to the UNL from the Negative UNL. This shares some similarities to the mechanisms in our framework, in particular, the tracking of nodal performance and mechanisms to weed out nonperforming nodes.

3.3 Tendermint

Tendermint [29] is a permissioned blockchain that uses a custom Tendermint Proof-Of-Stake BFT. Each validator in the network is identifiable by its public key, and a complete copy of the entire ledger is stored. The validators take turns proposing new blocks and voting on whether they should be added to the ledger. The consensus protocol requires that $n > 2/3$ of validators agree in terms of staked coins, where n is the number of validators participating in the consensus round. The consensus expects two rounds of voting for each addition of blocks and uses a locking system to ensure the safety of the system. At the beginning of each round, a new *proposer* is selected round-robin and gets a specific amount of locally measured time to propose a block. Should the proposer exceed this time, a new round is initiated with a new proposer. After this initial proposal step, the system is asynchronous in the sense that it will only make progress if more than two-thirds of validators agree to the block. Tendermint uses a specific governance module called Governmint to manage the membership of all validators. Management is enabled through the consensus, where an *EndBlock* message is used to allow users to vote on the validity of a validator by reducing the validator's voting power to zero. This on-chain governance is an important part of the Tendermint system in order to maintain a network of healthy validators.

3.4 Hyperledger fabric

Hyperledger Fabric [4] is a permissioned blockchain system designed to be highly modular and specifically aimed at enterprise solutions. Because of this, an essential mechanism in Hyperledger Fabric is membership management, which allows access to the system and enforces policy rules and membership roles. A central component in Hyperledger Fabric's membership management is the Membership Service Provider (MSP). The MSP operates by interfacing with a Certificate Authority (CA) trusted by the organizations participating in the system. The CA issues digital certificates that are stored with and used by the MSP to authenticate members. The role of the MSP expands even further than providing access control since it can be specifically configured to allow various levels of access to different components of the system, such as data and channels [30].

3.5 Membership management

Several papers have explored various methods for access control and membership management in permissioned and consortium blockchains. In [31], Galledörfer et al. propose

a method to streamline the addition and management of nodes using Transport Layer Security (TLS) certificates. The certificates are associated with publicly recognized domain names that are specifically allowed to enter the system via on-chain configuration. By storing X.509 certificates in a smart contract on the chain that mirrors relevant certificates from an existing Public Key Infrastructure (PKI), the chain can automatically verify incoming nodes, reducing manual intervention from the system administrators.

In [32], Ghosh et al. explore the challenge of identity management for blockchain interoperability. In the scope of multi-chain interoperability, they address the problem of the diverse identity management methods being used in different systems and propose a solution to this by developing distributed identity registries. The distributed identity registries are shared ledgers managed by trusted identity providers that store and maintain identities. Additionally, the protocol provides a dynamic membership discovery mechanism where different networks have access to each other's membership list with associated credentials, allowing for real-time information on which nodes are currently allowed to participate.

In [33], Psarra et al. describe a novel approach to dynamically allowing and restricting access to sensitive healthcare data. This proactive access control is specifically designed to allow dynamic access to Electronic Health Record (EHR) systems based on the patient status. The system uses the default Hyperledger Fabric membership management but employs a Long Short-Term Memory (LSTM) neural network on the patient's data to predict critical health metrics and potential emergencies requiring different users to have access to different parts of the system. The use of smart contracts automates the access control decisions based on the outcome of the model, ensuring prompt access when needed.

4 Framework for membership management

4.1 Membership authority

The focal point of our framework for Dynamic Membership Management is a Membership Authority (MA). The MA maintains a database of the nodes authorized to write to the blockchain. In this paper, we are only concerned with the writers, and for the rest of the section, we assume that all nodes in the network are writers. Although the MA could be decentralized and implemented on a blockchain, we assume that for any node, it appears as a single source service (e.g., identified by a URL), as depicted in Fig. 1. The MA operates asynchronously to the

underlying blockchain and typically on timescales several orders of magnitude higher.

The MA database is versioned in the sense that any change to the membership is a new version. At a minimum, the attributes for each node include: *name*, *public key*, *IP address*, *IP port*. The difference between version $t + 1$ and version t defines which nodes exit or enter the system. The database may contain additional attributes of performance, compliance, wallet, stake, penalty points, and, more generally, a reputation vector, but updates to such attributes do not constitute a version change. The MA sets the appropriate performance targets and thresholds.

A node can connect to the MA to request joining the chain as a permitted writer. Depending on the particular chain policy, the MA may require credentials, an entry fee, a bond, and more. Once admitted, the MA adds the new node to its list of allowed writers.

We contemplate that, in some cases, there will be a constant source of new entrants. Similarly, nodes may be removed from the blockchain and go back to waiting to be admitted again.

Figures 2 and 3 show how the MA receives a request from a node requesting to be added to the set of active writers.

The lifecycle of a writer in a permissioned blockchain managed by our MA is as follows:

1. A new writer requests to be admitted to the chain and enters a waiting queue.
2. After satisfying the admission criteria, it enters the (priority) queue of candidates.
3. Once a slot for active writers on the blockchain becomes available, the next node on the candidate queue is selected and added to the set of writers.
4. The node is an active writer until it i) voluntarily exits, ii) exceeds its term if one is defined, or iii) is removed for lack of contribution (whether non-performance, faults, or non-compliance), and enters an exit queue.
5. The node completes the exit protocol, if any (e.g., may involve recovering assets or securities put up as a bond/stake).

The candidates may immediately participate as observers in the network, although they do not become active writers (or validators) until activated. The Membership Authority may be decentralized itself, such as a Decentralized Autonomous Organization (DAO) on the Ethereum blockchain.

Communications between the active blockchain nodes and the MA:

- For the active nodes, the MA responds to queries related to the membership.

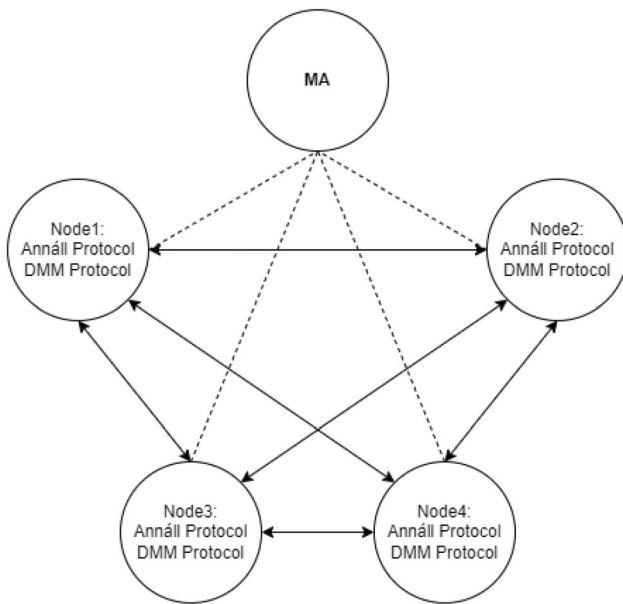


Fig. 1 Each node runs the membership protocol and has two-way access to the MA

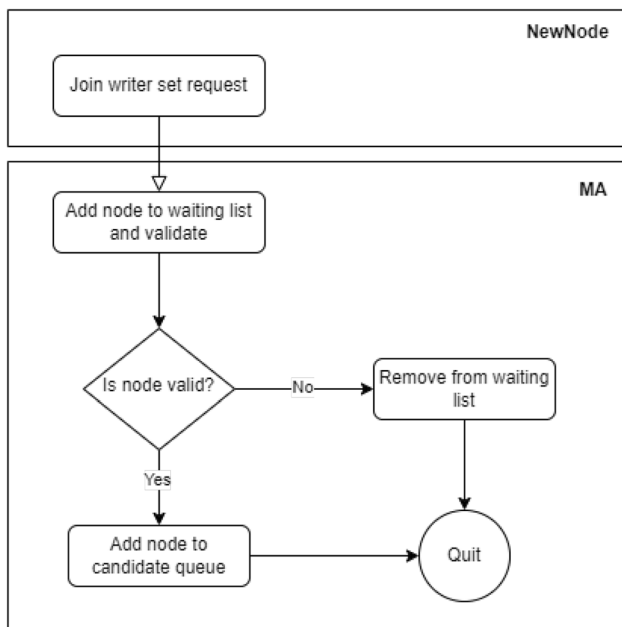


Fig. 2 The figure shows steps 1 and 2 in a writer's lifecycle depicted below. A node sends a request to the MA to become a writer and is admitted or rejected to a priority queue, depending on the node's validity

- The nodes publish reliability data related to other nodes, including votes against other nodes.
- The MA monitors the blockchain, thus observing which nodes are contributing to the construction of the blockchain.
- To exit the chain.

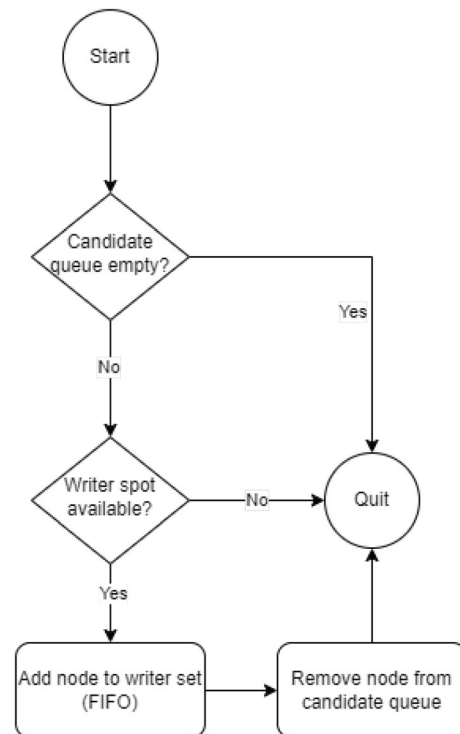


Fig. 3 The figure shows the process of step 3 in a writer's lifecycle depicted above. The MA repeatedly tries to add previously validated nodes to the active writer set

Every writer maintains a local view of the set of writers and the MA version it is operating under at any time. Additionally, it maintains online status for each node, where status is either *active*, *penalty box*, *entering*, or *exiting*.

The MA actively monitors the progression of the blockchain for biases, collusion, and other misbehavior, ruling out misbehaving nodes.

4.2 Entry—addition of a new member

Each active node publishes its membership list with its status on every round (to a subset of nodes), as detailed further below. Such publication includes the currently active version and the next version if one exists. Each node occasionally contacts the MA to see if the version has changed, in which case it retrieves the new one and starts publishing this new version as the next one. It locally calculates the difference between the two versions.

Anyone receiving a message that has a next version greater than the current one is thereby informed that a later version exists and retrieves it. Once sufficiently many have received the next version, the system moves to make the next version the current.

A newly added node must initially retrieve a copy of the entire blockchain or request missing blocks, should the node already contain some of the previous blocks.

Depending on the chain, the node may be able to participate in the construction going forward before this “catch-up” is fully completed. The details and policies on this are orthogonal to this work.

4.2.1 Bootstrapping

Each node in the permitted chain bootstraps by retrieving the current version of the permitted writer set from the MA and starts connecting to the other nodes in the network. If the chain is starting, i.e., the node belongs to the first batch, once connected, it prepares to participate in the next round and awaits the initial round startup. The first round may be initiated by a designated node, e.g., the first on the list, or be preceded by another leader election process. We assume the former.

4.3 Exit—removing a member node

Writers leave the chain, either voluntarily or are removed at the initiative of the MA. To leave gracefully and voluntarily, the node sends a *request to exit* message to the MA, which removes the node from the list of allowed writers. The exiting node continues to operate normally while the MA circulates an updated allowed writers list, with the exiting node removed. Upon receiving this list, the exiting node can confirm that it has been removed from the set of writers and disconnects from the system.

Writers not departing at their own initiative are removed from the chain by the MA. We do not assume as to what policies govern the removal of a node from the network. Such policies may include confiscating stakes or other penalties, simple policies such as limited terms, and more.

However, in addition to the MA potentially monitoring the contribution of the nodes in the system, our framework includes feedback from the members. Specifically, the protocol allows nodes to report complaints, essentially proposing that some nodes be removed.²

4.4 Timescales

In distributed systems, particularly peer-to-peer blockchains, it is important to recognize the distinct timescales at which various processes operate. Three key timescales can be identified:

Blockchain Lifetime represents the longest timescale, encompassing the entire operational span of the blockchain. It involves major changes to governing protocols, such as the adoption of a new consensus algorithm or substantial network upgrades. Such transformations occur

infrequently, often over the course of months or years, as they necessitate broad consensus and a coordinated implementation effort from the network participants.

Changes in Membership pertains to the modification of the set of participants who contribute to the construction of the blockchain. This timescale includes the addition or removal of validators in permissioned systems. Changes in membership occur on a slower timescale compared to block production, typically one or two orders of magnitude less frequent. This timescale reflects the dynamic, yet relatively stable, participation in the network.

Block Construction and Consensus timescale concerns the rate at which new blocks are appended to the blockchain. This process advances incrementally, with each round producing a new block. Block creation intervals are comparatively short and are typically fundamental to the system’s design.

While block generation occurs at the fastest rate, alterations in the set of participants take place at a slower pace, and fundamental changes to the governance of the blockchain unfold over the longest period. Our membership management framework operates at the intermediate timescale, with interactions with the block construction processes below.

5 Annáll—an elementary blockchain system

To prototype and experiment with our membership management framework, we have implemented it using our own elementary blockchain system, Annáll, proposed and described in [34]. For completeness, this section describes the main mechanisms of this elementary blockchain system. First, an elementary event log is built on a peer-to-peer time-stamping service. Second, a lightweight decentralized consensus protocol, where we briefly address the correctness and complexity of the protocol. Lastly, we discuss the management and monitoring of the group of writers.

This paper does not propose Annáll as a competitive blockchain or as a major contribution. Instead, Annáll serves as an elementary and generic time-stamping blockchain that we use as a workbench for research and experimentation. In this context, Annáll is utilized as the blockchain on which we prototype our Membership Management Framework. That notwithstanding, we have used Annáll extensively, operating an Annáll blockchain for months on end, and have built a number of interesting services on it.

From the Membership Management Framework, Annáll merely serves as a protocol blockchain and could be replaced by more established blockchains such as Ethereum, XRPL, or Tendermint, though doing so would be

² Note that even though nodes may complain to the MA, different policies will direct how the MA deals with such complaints.

more convoluted and require significantly more development effort. Therefore, Annáll is primarily used as a tool for prototyping and experimentation with new mechanisms.

5.1 A simple lightweight event log

Our event log is a timestamp server, similar to that described in [35] and adopted for Bitcoin. No service is provided beyond entering the payload into a block on the chain. The system progresses in rounds, each round organized by a coordinator who orchestrates the winner selection but is ineligible as a winner during that same round. Winners are selected randomly, as detailed below. The winner proposes and distributes the next block if all steps are successful. Otherwise, the round fails, no block is produced, and one or more nodes may be penalized.

Each entry in the log consists of:

- Coordinator ID: The ID of the coordinator for the round the block was created in.
- Winner ID: The ID of the round winner and creator of the block.
- Body: An arbitrary payload.
- Hash: The hash of the block.
- Previous Hash: The hash of the previous block.
- Winner Signature: The signature of the blocks creator.

As with other blockchains, the inclusion of the hash of the previous block in the hash of the new block, by induction, provides the immutability of the chain with respect to the set of writers. Annáll makes no assumptions about the body, which may be composed of multiple transactions of arbitrary data and may be further organized, e.g., into Merkle trees [36].

The chain is immutable, as no previous block can be changed. Each block is cryptographically signed by the winner who writes it.

5.2 A lightweight consensus

For a high-level view of the underlying idea of the lightweight decentralized consensus protocol, we select a leader at random based on the following. Each writer has a private, cryptographically secure One-Time Pad (OTP) [37]. In each round, each writer produces the next number from its OTP. From all the numbers, we calculate an aggregate result, i.e., a generalized *sum*, used to select the winner.

5.2.1 The consensus protocol

To formalize the above idea into a protocol, we define W as a set of the n writers and start by considering a single round. First, we enumerate the writers set from 0 to $n - 1$.

To coordinate the round, a *coordinator* is selected for each round. The Coordinator does not participate in the round as a contender to win but orchestrates the round as a service to the peer-to-peer network. For the initial round, the writer with the lowest number acts as the coordinator. We use a simple round-robin to select the round's coordinator for subsequent rounds.

The round progresses as three separate Remote Procedure Calls (RPC) and is fully broken down into the following six steps:

- Step 0a: The Coordinator initiates the round by broadcasting to all a *prepare* message.
- Step 0b: Each writer generates the next number r_i from its local OTP, computes $h_i = \text{hash}(r_i)$ and transmits a *prelude* message containing h_i plus the hash of the previous block h_{prev} to the Coordinator.
- Step 1: The Coordinator broadcasts to all a *request* for the next number, including in the message $\vec{h}$ containing the hashes received from each node.
- Step 2: Each writer computes the hash h of its proposed next block and *responds* by transmitting its next number r_i plus the hash h_{prev} of the previous round to the Coordinator.
- Step 3: The Coordinator collects a vector $\vec{r}$, initialized to zero, where r_i is the number received from node i if any, and verifies that $h_i = \text{hash}(r_i)$ for all i . The response from node i is only accepted if h_{prev} is correct. It then aggregates the numbers into a result r and determines the winner, $w = r \bmod n$. The Coordinator then broadcasts a *result* message to all writers in W , namely the vector $\vec{r}$, the winner w , and the hash of the next block h_w .
- Step 4: Each writer verifies the round by confirming i) that its number is correctly received and reported, ii) $h_i = \text{hash}(r_i)$, iii) the calculation of r , and thus iv) the winner w . The winner broadcasts its winning block. All nodes receiving the new block verify its hash matching h . All nodes for which all tests succeed reply with a *confirm* message to the coordinator and the next coordinator such message including the pair (w, h_w) .

Protocol messages are sent using TCP connections, with each node connected directly to up to a few hundred nodes. The sequence of operations during a normal case execution of a single round is shown in Fig. 4. The praround Step 0a and Step 0b are omitted for clarity, as they are identical to the request and response. We further give an example in Listing 1 on a single message as it is constructed and parsed by the network.

The round is completed, and the next round proceeds once sufficiently many $(n - f)$ confirms are received by the next coordinator.

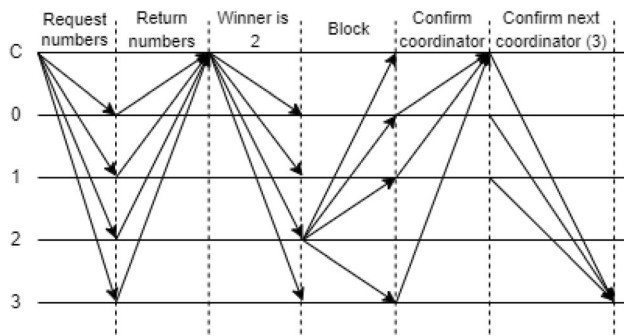


Fig. 4 The sequence of operations during a normal case execution of a single round. The pround Step 0a and Step 0b are omitted for clarity, as they are identical to the request and response

Listing 1: An example of a message object and aparsed object as it is sent out to the network

```
#Message class
class Message:
    def __init__(self, current_round,
                 from_id, to_id, msg_type, msg):
        self.c_round = current_round
        self.from_id = from_id
        self.to_id = to_id
        self.m_type = msg_type
        self.msg = msg

    def _send_msg(self, message):
        #<round nr>
        #<from id>
        #<to id>
        #<msg type>
        #<msg body>
        parsed_msg = parse(message)
        self.comm.send_msg(
            parsed_msg)
```

A number of alternatives exist to aggregate the number from all the writers. For its simplicity when working with large numbers (e.g., multiple words) and for its cryptographic properties, we have chosen to use XOR as the aggregation function.

5.3 Protocol complexity

As with every blockchain system, the progression of the construction is self-clocked by the conclusion of consensus and the addition of the next block. In Annál, we observe that any node sending a valid response in Step 2 includes

the hash of the previous round, thereby implicitly acknowledging the chain up to that block. Moreover, in Step 3, the Coordinator sends the result of the round, including the $\vec{r}$. Any node receiving the result can rightfully conclude that any node i , for which $r_i \neq 0$, is participating in this new round and thereby confirms the prior round.

The message complexity for the protocol is as follows: In Step 0a, the coordinator sends $n - 1$ messages of fixed size, which each of the $n - 1$ writers responds to with one message. In Step 1 the coordinator sends $n - 1$ messages, each of size $O(n)$, to which each of the $n - 1$ writers responds with one message. For Step 3, the coordinator sends $n - 1$ messages, one to each writer, each of size $O(n)$. For step four, all but the coordinator and the winner send a confirm message to the coordinator and the next coordinator for a total of $n - 2$ messages of fixed size. Lastly, the winner sends $n - 1$ messages to each of the writers to transmit the new block to all.³ Hence, the number of messages to complete a round is $O(n)$.

Message size is dominated by the pre-step Step 0b and Step 3, where the coordinator sends the vectors $\vec{h}, \vec{r}$, respectively. Hence, the size of the message is $O(n)$. Other messages are of constant size and are small. In general, message size of $O(n)$ is an issue for sufficiently large networks. Still, as a trade-off against $O(n^2)$ messages, the linear message size seems efficient. For the numbers we envision for the writer sets in permitted blockchains, this trade-off seems appropriate.

Each round of the protocol is essentially three one-to-many RPCs, each completed in one message exchange. Each message exchange consists of a transmission, followed by processing at the remote end, followed by a transmission back. In Step 0b, and Step 3, for large n the analysis might include the impact of the $O(n)$ message. For geographically distributed writer sets, latency during consensus will be dominated by the propagation delay of the maximum inter-nodal distance in the peer-to-peer network. For very localized setups where interrupts and the protocol processing become the dominant factor, however, the absolute values of the latency are insignificant.

The above is based on the assumption that a round is completed before the next can commence. However, rounds can be overlapped, allowing the next round to start before the current one is completed. For the work herein, this is not crucial.

³ The block may have been previously sent, e.g., via gossip protocol. Whereas doing so makes the propagation of the block asynchronous to the consensus, it does not reduce the number of messages that need to be sent.

5.3.1 Local complexity

Due to its elementary service model, the local complexity of an Annáll writer is small. It is composed of three components: state related to the membership management, state related to the peer-to-peer network connectivity, and state related to consensus. In contrast to other blockchains, no service-related state, such as wallets, is maintained.

The necessary state related to membership management consists of version information and the list of writers in the permitted chain, with static information for each writer, mainly writer ID, public key, and remote end information. Clearly, this state is of the order of $O(n)$, and changes on the timescale of membership changes.

Maintaining connectivity and otherwise participating in the peer-to-peer network requires maintaining connectivity and routing and forwarding messages. This includes maintaining a relatively small constant number of timers and determining the next hop routing information for each writer. The routing state is a fixed amount of information per writer. The resulting state is $O(n)$, with changes on the timescale of several consensus rounds.

The state related to consensus is the most critical and challenging part of it, as it is inherently shared. For the current round, the writer keeps track of the round number, the current version in use, the current and next coordinator, and the vector $\vec{r}$ of the responses from each writer. In addition, it keeps two vectors of size n , one containing the penalty box for the round and another containing the penalty box information as computed locally. The writer may keep a history of the last k rounds to track the participation and reliability of all n members. Clearly, the space complexity of this is $O(n)$.

Perhaps more important than the size is to appreciate the timing and synchrony of the consensus state. The round number, the current version, and the round exclusions (those nodes not taking part in this round) are provided by the coordinator. If either the number of the version received differs from the local state, the node will not participate in the round. The round exclusions are dictated by the coordinator and need not be consistent with the local view. Hence, although sufficient information is shared to allow all nodes to keep track of the reliability of their peers, it is unnecessary for that state to be fully consistent.

We conclude that the overall local complexity is $O(n)$. Even for very large memberships, this state is dwarfed by the service state, such as that of wallets, in blockchains implementing richer service models.

5.4 Correctness

The purpose of the consensus algorithm is, in each round, to select the writer w that becomes the winner, i.e., gets to

write to the chain. Each process draws an initial value from its OTP. The goal of the protocol is that all valid writers decide on the same value w . The problem can be formally defined in terms of three properties:

- *Validity*: If all correct writers propose the same value w , then any correct process that decides, decides w .
- *Agreement*: No two correct writers decide differently.
- *Termination*: Every correct process eventually decides.

The first two properties ensure correctness, whereas the third ensures termination [38].

The correctness follows from the fact that each node receives the vectors $\vec{h}$ and $\vec{r}$ and therefore independently verifies that h_i is the hash of r_i and computes the result w . Moreover, the winner, w , sends the block whose hash h is again independently verified by every node.

There is no potential for Byzantine failures in this system. The only use of broadcast is from the coordinator. Since the hashes are sent separately before the numbers $\vec{r}$, there is no opportunity for the coordinator to send one set of values to one subset of nodes and a different set to the remaining nodes without it being detected.

By definition, a node not responding within a certain maximum Δt will time out and be deemed failed. It follows that every correct process will respond within that time limit and thus decide.

6 Monitoring and removing misbehaving writers

In this section, we discuss how active writers monitor the reliability of each other, isolate and penalize non-performing writers, and may ultimately remove writers deemed too unreliable. This enforcement is aimed at minimizing the impact of failing writers and maintaining a standard of performance and contribution to the peer-to-peer network. Removing unreliable writers is essentially a garbage collection to avoid having a growing number of noncontributing writers in the blockchain. In particular, we allow for policies that remove offenders from entering the penalty box too frequently.

6.1 Version maintenance

Each writer in the network keeps track of the membership versions, starting with an initial version obtained during the bootstrapping process from the MA. At any given time, a writer participates in the consensus protocol using its current version. Additionally, a writer may become aware of a newer version, which it retrieves and prepares for. Once ready, the writer is able to migrate to the updated version.

Writers can learn about newer versions either from other writers during the blockchain construction process or by periodically querying the MA for the latest version. Upon discovering that a new version is available, the writer retrieves it and prepares to migrate by establishing connectivity with any new nodes introduced in the updated version.

For each protocol message it sends, a writer includes both its current version number and the latest version it is prepared to migrate to. This ensures that other writers become aware of the version the sender is operating on, as well as any impending upgrades. When receiving protocol messages, the writer similarly learns the sender's version information, facilitating coordination during version transitions.

Each writer maintains a state for every writer in the writing set. This state includes the writer's ID and public key, as well as the protocol penalty state of the writer, counting from zero.

The connectivity of the peer-to-peer network connecting the group of writers is maintained separately from the consensus protocol. At the protocol level, disconnected nodes are treated as being in the penalty box.

6.2 Maintaining the penalty box

To isolate and minimize the impact of nonperforming writers, we propose a concept of a *penalty box*, where misbehaving writers are temporarily quarantined from participation for one or more rounds.

Each writer locally maintains a performance history for each member of the network. In each round, every active node that receives the result message in Step 3 of the protocol implicitly receives information about the active writer set. Namely, every non-zero entry of $\vec{r}_t$ indicates an active node in round t . Conversely, the zero entries communicate which nodes are faulty. Storing a history of the last k rounds allows the node to compute a reliability metric for each peer.

The penalty box is managed as follows. At the start of a new round, the coordinator reviews the reliability of all nodes. For a node with the poorest recent performance, as measured by failed rounds, the coordinator may decide to exclude it from participation by adding it to the penalty box. The penalty box is communicated to all participants in the coordinator's initial request message. Nodes in the penalty box are effectively ignored in the consensus calculation for that round.

Each entry in the penalty box includes attributes of severity and release time set to the round where the node will participate again in the consensus. The severity is initialized at four rounds and increases exponentially with subsequent penalization. The maximum size of the penalty

box is provided by the MA. Other nodes receiving the penalty box from the coordinator update their local penalty box to match.

The intent of the penalty box is to reduce the number of failure events that affect the consensus. Nodes that fail and recover only after many rounds would otherwise be seen as failing in every round, effectively using up part of the fail quota afforded by f . However, by quarantining them, n has effectively been reduced, allowing us to treat f purely to capture temporally uncorrelated random failures. This allows f to be set at a small number.

Whereas the penalty box is a shared state, it is maintained only lazily and may be divergent on different nodes without significantly impacting its effectiveness.

6.3 Removing non-contributing nodes

Occasionally, the coordinator may initiate a member cleanup round instead of starting the next ordinary round. During this round, the coordinator reviews the performance of all members and assesses whether any node's performance has degraded to the point where removal from the writer set is justified. Clearly, nodes of significant severity in the penalty box are prime candidates. The performance thresholds for this evaluation are provided by the MA. If a node is identified for potential removal, the coordinator initiates a remove member round, which includes the ID of the member in question. This process follows a structure similar to the consensus round. Each participating node casts a vote, returning a random number if voting in favor of removal or zero if voting against it. If the number of favorable votes exceeds the threshold, the node is removed from the writer set and the MA is notified of the same.

7 Analysis and simulations

One of the significant challenges in studying the performance of blockchain systems is the lack of performance data, which impedes the validation of assumptions for connectivity and failures. We note that a receiving node (either timing out or receiving a message that cannot be validated), in most cases, cannot determine if a failure is caused by a communication error or due to processing at the remote node. In each round, our consensus algorithm performs three message exchanges, each effectively an RPC, which either succeeds or fails, regardless of the reason.

Although the source of failure may be indistinguishable, two distinct types of failures affect the system. On one hand, nodes may fail and/or go offline entirely. A node experiencing such a failure typically remains unavailable for multiple rounds, potentially for minutes or even hours.

On the other hand, message exchanges may fail due to various random factors, causing node i to fail in round t without affecting its ability to participate in the subsequent round. Most research on BFT and consensus protocols in blockchain systems, however, focuses on the latter case, assuming temporally independent random failures of nodes. Classical BFT analyses are typically concerned with achieving consensus in a single round. In this memoryless failure model, a random subset of nodes fails in each round, but consensus is reached as long as fewer than one-third of the nodes fail.

Below, we provide an analysis of each type of failure. First, a memoryless failure model is assumed, capturing failures from random factors. Second, we use a two-phased model where nodes are either active or failed and where a failed node stays failed for an appreciable number of rounds. In both cases, all nodes are identical and fail independently. A round is successful if $s > n - f$ nodes successfully complete the rounds, where $f < n/3$ is a given fault tolerance.

7.1 Memoryless failures

Assuming memoryless failures in the system, as well as symmetric and independent failure characteristics for all nodes, the system can be modeled as a series of Bernoulli trials. Let the probability of a round succeeding be p , and failing $q = 1 - p$.

Considering the progression of each node as a random walk, let $X_i(t)$ represent the state (failure or success) of node i in round t , defining $X_i(0) = 0$. At each round t , the node i either fails or responds with the state transition: $X_i(t+1) = X_i(t) + S_i(t)$, where $S_i(t)$ takes the value -1 if successful, and $+1$ if failed. We conclude that the expectation and variance for $S_i(t)$ is given by:

$$E[S_i(t)] = q - p, \text{ and } \text{Var}[S_i(t)] = 4pq$$

from which it follows that $E[X_i(t)] = j \cdot (2q - 1)$, and $\text{Var}[X_i(t)] = j \cdot 4pq$.

The total number of nodes that fail in round t , is a binomial random variable given by $\phi(t) \equiv \text{Binomial}(n, q)$, with expectation and variance given as

$$E[\phi(\cdot)] = nq \text{ and } \text{Var}[\phi(\cdot)] = npq.$$

Similarly, using $C_i(t)$ as the cumulative number of failures for node i up to and including round t , we get that $E[C_i(t)] = tq$ and $\text{Var}[C_i(t)] = tpq$.

Using this model, the sojourn time that a node spends as failed is geometric, with the likelihood of k failures given by pq^{k-1} , with expectation $1/p$ and variance q/p^2 .

A consensus is reached, and a winner is declared in a given round iff $k \geq n - f$ nodes complete the round successfully. This is the tail of the Binomial(n, p) distribution given by

$$P(X > k) = 1 - \sum_{i=0}^k \binom{n}{i} p^i q^{n-i}$$

7.2 Modelling of failed writers

To better model nodal failures, where a node goes offline for a number of rounds, we use a two-phased *Modulated Markov Process* (MMP). In this model, each writer is considered to be in an active state until it fails (for whatever reason), and moves to a failed state for some time, until it recovers and returns to being active. We assume that nodes fail independently.

With $state = 0$ denoting active and $state = 1$ denoting failed, the transition matrix of this two-phase on-off model is

$$P = \begin{pmatrix} p_{00} & p_{01} \\ p_{10} & p_{11} \end{pmatrix}, \sum_{j=0}^1 p_{ij} = 1, i, j = 0, 1$$

It follows that the steady-state probability that the system is in $state = i$ is

$$\mu = \pi_0 = \frac{p_{10}}{p_{10} + p_{01}}, \pi_1 = \frac{p_{01}}{p_{10} + p_{01}}$$

Hence, the probability that a given writer is active is μ . The steady-state distribution of nodes in an active state at the end of round t is given by $\varphi(t) = \text{Binomial}(n, \mu)$.

The expected sojourn time as in $state = i$ is $E_i\{L\} = 1/(1 - p_{ii})$, with variance $\text{Var}_i\{L\} = p_{ii}/(1 - p_{ii})^2$. Given average up-time μ , and average failure duration in rounds τ , we get $p_{10} = 1/\tau$, and $p_{01} = (1/\mu - 1) \cdot 1/\tau$.

7.3 Simulations

In this subsection, we show simulation results for effective throughput (number of completed rounds vs. failed), as the controlling parameters vary. We show how the number of failed rounds is reduced by using the penalty box. Lastly, we will determine how frequently the peer-to-peer network will take action to remove a member from the active writer set.

As a base reference case, we use the memoryless model. We compare that to the results of the two-phase model and then continue with the two-phase model to simulate the system with a penalty box and membership management. For all simulations, the key parameters are i) n , the size of the writer set, ii) f , the threshold of failed nodes for a round

to complete, iii) μ , and the expected up-time of writers. When using the two-phase model, we add iv) τ , the duration of failures.

7.4 Simulation setup

The simulation setup is as follows. The writers are initialized. In each round, each node runs an experiment to determine if it successfully completes the round and returns a value to the coordinator. A consensus is reached, and a winner is declared in a given round if $s \geq n - f$ nodes complete the round successfully. Otherwise, the round fails. If using the penalty box, a node that fails enters the penalty box, where it stays for a number of rounds before returning to the protocol. Using this setup, we run a number of experiments.

We simulate for the number of writers n , in the range $20 \leq n \leq 400$. All nodes are identically configured in terms of expected up-time and duration of failures. Varying the duration of failures captures whether failures are frequent but minor or fewer but have failure times reflective of requiring human interventions. The latter will occur on timescales of minutes, hours, or days. For the purposes of these simulations, we assume one round per second.

7.5 Simulation results

As a base case for our simulation studies, we use the memoryless model discussed in Sect. 7.1, where each round is a series of Bernoulli trials, each node failing independently and without memory of prior round(s). In each run of the simulation, we set the expected up-time, μ , to a fixed value identical for all nodes. Each simulation is run for 1,000,000 rounds.

Table 2 presents the success rates for various values of f in a network comprising 40 nodes, assuming memoryless failures, as the average node uptime, μ , is varied. In this configuration, the expected number of failures per round is 8 for $\mu = 80\%$ and 6 for $\mu = 85\%$. Thus, it is unsurprising that at $f = 6$, the success rates remain low. We note that for

40 nodes, the maximum tolerance satisfying $f < n/3$ is $f = 13$, and this is independent of μ .

In the table, the values for f where the system's overall success rate matches or exceeds the expected success rate of each individual node are highlighted. We observe these highlighted values at $f = 6, 4$, and 3, and for $\mu = 90\%$, 95%, and 98%. These results underscore the key insight that as the reliability of individual nodes improves, the overall system performance benefits correspondingly. This evidence suggests the importance of actively managing the network of nodes to ensure their sustained reliability and contribution to the system's success.

While this may seem intuitive, a review of the literature reveals a community that is disproportionately focused on mitigating node failures and accommodating unreliable nodes, often at the expense of advocating for active management and access control mechanisms.

Table 3 shows the success rate for different values of f , with each node having reliability $\mu = 95\%$ as the number of nodes in the network, n , is varied. In this model, it is clear that we need $f > nq$. Again, we highlight the values for f where the expected success rate of the entire system matches or exceeds the expected success rate of each individual node, in this case, 95%. We note that in all cases, this occurs for values of f significantly below the tolerance satisfying $f < n/3$.

7.5.1 Using the two-phase model

To establish a basis for our two-phase model, we run a set of simulations configured in exactly the same way as with our memoryless simulations, except with the additional parameter τ , denoting the duration of failures, starting with $\tau = 100$ rounds.

Table 4 provides the success rate for different values of f , with each node having reliability $\mu = 95\%$ as the number of nodes in the network, n , is varied, using the two-phase model. The value for τ in this case is 100. As evident, the table is almost identical to Table 3.

Although the two-phase model exhibits higher variance, the success rate follows a $\text{Binomial}(n, \mu)$ distribution. While the duration of failures affects the transient behavior of the system, the steady-state long-term success rate remains equivalent to that of the memoryless model. Thus, the steady-state success rate is independent of τ . However, as will be demonstrated below, the value of τ has a significant impact on the failure dynamics of the system.

7.5.2 Adding the penalty box

We analyze Tables 5 and 6 in tandem, as they are presented for comparative purposes. Table 5 displays the results of simulating the two-phase model without the penalty box,

Table 2 The success rate for different values of f in a network of 40 nodes, using memoryless failures, as the average up-time of nodes, μ , is varied

μ	$f = 2$	$f = 3$	$f = 4$	$f = 5$	$f = 6$
80%	0.8%	2.8%	7.5%	16.1%	28.6%
85%	4.8%	13.0%	26.4%	43.2%	60.6%
90%	22.3%	42.3%	62.9%	79.3%	90.0%
95%	67.7%	86.2%	95.2%	98.6%	99.7%
98%	95.4%	99.2%	99.9%	100%	100%
99%	99.2%	99.9%	100%	100%	100%

Table 3 The success rate for different values of f with each node having reliability $\mu = 95\%$ as the number of nodes in the network, n , is varied

n	$f = 0$	$f = 1$	$f = 2$	$f = 3$	$f = 4$	$f = 5$	$f = 6$	$f = 7$	$f = 8$	$f = 9$
20	35.8%	73.6%	92.5%	98.4%	99.7%	100%	100%	100%	100%	100%
40	12.8%	39.9%	67.7%	86.2%	95.2%	98.6%	99.7%	99.9%	100%	100%
100	0.6%	3.7%	11.8%	25.8%	43.6%	61.6%	76.6%	87.2%	93.7%	97.2%
200	0%	0%	0.2%	0.9%	2.6%	6.2%	12.4%	21.3%	32.7%	45.5%

Table 4 The success rate for different values of f with each node having reliability $\mu = 95\%$ as the number of nodes in the network, n , is varied, using the two-phased model, and $\tau = 100$

n	$f = 0$	$f = 1$	$f = 2$	$f = 3$	$f = 4$	$f = 5$	$f = 6$	$f = 7$	$f = 8$	$f = 9$
20	36.2%	73.8%	92.4%	98.3%	99.7%	100%	100%	100%	100%	100%
40	12.5%	39.6%	67.6%	86.7%	95.2%	98.3%	99.6%	99.9%	100%	100%
100	0.6%	3.0%	11.6%	25.7%	44.2%	60.1%	76.5%	87.9%	93.1%	97.6%
200	0%	0%	0.2%	1%	2.4%	5.9%	12.2%	21.2%	32.2%	45.8%

Table 5 Two phase model: The success rate for different values of f with each node having reliability $\mu = 95\%$, and $\tau = 600$, as the number of nodes in the network, n , is varied

n	$f = 0$	$f = 1$	$f = 2$	$f = 3$	$f = 4$	$f = 5$	$f = 6$
20	35.7%	73%	93.6%	98.5%	99.7%	100%	100%
40	12.0%	37.4%	68.8%	86.2%	95.1%	98.3%	99.8%
100	0.6%	3.9%	10.3%	25.9%	44.3%	63.5%	77.5%
200	0%	0.1%	0.2%	1%	3%	5.8%	13%

Table 6 Two phase model with penalty box: The success rate for different values of f with each node having reliability $\mu = 95\%$, and $\tau = 600$, as the number of nodes in the network, n , is varied

n	$f = 0$	$f = 1$	$f = 2$	$f = 3$	$f = 4$	$f = 5$	$f = 6$
20	96.9%	100%	100%	100%	100%	100%	100%
40	93.8%	99.8%	100%	100%	100%	100%	100%
100	85.2%	98.9%	100%	100%	100%	100%	100%
200	71.7%	95.5%	99.5%	100%	100%	100%	100%

while Table 6 presents the results with the penalty box mechanism. Both tables report the success rates for various values of f , assuming a node reliability of $\mu = 95\%$ and $\tau = 600$, as the number of nodes in the network, n , is varied. Further, for the simulations shown in Table 6, the average occupancy of the penalty box for each network size of 20, 40, 100, 200 is noted to be approximately 1.5, 3.2, 7.6 and 15.6 respectively.

In these simulations, the penalty box is configured such that a node experiencing failure is placed into the penalty

box for τ rounds, after which it re-enters the pool of active nodes.

The comparison between these tables reveals a striking difference. The effect of the penalty box mechanism is profound and cannot be overlooked. Without the penalty box, we observe that f significantly exceeds nq , whereas, with the penalty box in place, the value of f remains relatively stable. This stark contrast highlights the critical role the simple mechanisms of the penalty box play in improving system robustness and managing node failures effectively.

Table 7 Two-phase model with penalty box: The success rate for different values of f in a network of 40 nodes, each node having long-term reliability $\mu = 95\%$ as the average duration of failures τ is varied

τ	$f = 0$	$f = 1$	$f = 2$	$f = 3$
100	83.9%	99.1%	99.9%	100%
300	88.3%	99.3%	100.%	100%
600	93.8%	99.8%	100.%	100%
900	95.7%	100%	100%	100%

Table 8 Two-phase model with penalty box: The success rate for different values of f in a network of 100 nodes, each node having long-term reliability $\mu = 95\%$ as the average duration of failures τ is varied

τ	$f = 0$	$f = 1$	$f = 2$	$f = 3$
100	68.5%	99.1%	99.9%	100%
300	72.9%	95.8%	99.5%	100%
600	85.2%	98.9%	100%	100%
900	89.6%	99.7%	100%	100%

To understand the impact of the duration of failure, τ , we present results in two network sizes, as τ takes on increasing values. Table 7 presents the success rate for different values of f in a network of 40 nodes, each node having long-term reliability $\mu = 95\%$ as the average duration of failures τ is varied. Similarly, Table 8 shows the same for a network of 100 nodes. There is a small impact of the larger member size in that with 100 nodes, we need f larger by one or two.

To ground these values for failure duration, at one round per second, the failure duration values 100, 300, 600, and 900 correspond to 2, 5, 10, and 15 min, which seem on the lower side. Of course, if rounds take longer, e.g., 10 s, these would correspond to 17 min to two and a half hours.

Across all timescales, we observe that as the duration of failures, denoted by τ , increases, the impact of the penalty box mechanism becomes more pronounced. This outcome is expected. For a given fixed uptime, longer failure durations result in fewer individual failures, each persisting for an extended period. In other words, compared to the memoryless failure model, the overall number of failures in the system remains unchanged; however, these failures are concentrated among a smaller number of nodes, each experiencing prolonged downtime. Our results show that isolating such nodes proves to be highly effective in mitigating their impact on system performance.

Our analysis and simulations further demonstrate that the use of an elementary mechanism to isolate failing nodes yields substantial improvements in overall system performance with minimal associated costs.

8 Discussion

In our analysis and simulations, we have used two models that allow us to study the failure behavior of the system and the potential benefit from our mechanisms. This study is, however, limited due to the lack of performance data, on the one hand, to set the appropriate parameter values and, on the other hand, to validate the underlying assumptions and outcomes.

We have been involved with some of the large data centers operating in Iceland, whose operations for some time include operating blockchain writers (including miners) in the thousands. Privately shared information indicates that in practice, nodal failures are rare and correlated, often to external adverse events. Getting access to such data would be critical in advancing the performance evaluation of blockchain systems. Additionally, we have prototyped and extensively experimented with the Membership Management framework and the Annáll lightweight blockchain. On top of this, we have developed higher-level services, including wallet service, life insurance service, digital certification, and more, using the blockchain system. Our experimentation includes running the Annáll blockchain and its accompanying dynamic membership management system for approximately nine months using geographically separated nodes.

While many permissioned blockchains continue to be managed by a central authority - typically an individual or a small team - our work aligns with the growing interest in adopting algorithmic membership management. Whether for automation, transparency, self-governance, or further decentralization, we see potential in experimenting with permissioned chains established and operated by DAOs. Whether existing on the chain itself or a separate blockchain, these DAOs would be governing access, compliance policies, and other aspects of the membership authority.

The prospect of recording membership management events directly on the blockchain is appealing. In this approach, management events, for example, addition and removals of nodes, would be written into the blockchain, from where the membership would be accessed by all. Although this has not been implemented in our prototyping, the decoupling of the MA from the blockchain core construction in our framework would certainly support such a system in the future.

Although our proposed framework and simulations demonstrate the potential benefits of dynamic membership

management in permissioned blockchains, several limitations remain. First, whereas our prototyping and simulations have demonstrated improved performance, such simulations do not encounter some of the adversarial scenarios that may occur in real-world large-scale blockchain environments. Second, in this paper, we primarily discuss mechanisms, with the only policies presented being the penalty box and its potential benefit. The policies governing the various mechanisms, for example, the node entry and exit policy, are a subject of further studies. Finally, although, in principle, our framework and the mechanisms are applicable to arbitrary-size networks, additional work is warranted on how the size of the network affects the quality and efficiency. We see this point as applying to blockchain systems in general.

9 Conclusion

In this paper, we have introduced a novel and equitable dynamic membership management framework for permissioned blockchains aimed at providing a structured approach for adding, removing, and monitoring members. This programmatic framework provides a loose coupling between the management of allowed writers and the consensus in use, allowing for a more flexible and efficient governance model and, as such, encapsulates a range of permissioned blockchains. The framework is designed not to interfere with the blockchain consensus and construction process and dynamically adjusts the active writer set, isolating and ultimately removing malfunctioning writers from the network.

By introducing the penalty box, we have shown how active monitoring and dynamic membership management can improve the overall performance of the system. Our simulation results highlight the significant benefits of ensuring high reliability among individual nodes, reinforcing the need for active monitoring and management within a permissioned network. These findings suggest that implementing appropriate incentives to maintain reliability and independence may be more effective than the current common practice of rewarding nodes based on the lottery outcome of being selected as the winner in a specific round. Our analysis and simulations further demonstrate that the use of an elementary mechanism to isolate failing nodes yields substantial improvements in overall system performance with minimal associated costs.

In conclusion, dynamic membership management is a critical component of any permissioned blockchain system. Our proposed framework provides a structured approach to improve the governance, reliability, and efficiency of those systems and can be used as a basis for further research in this particular area. By addressing the challenges noted

before, this paper contributes to the broader discourse on optimizing and structuring blockchain components in permissioned enterprise applications and solutions.

Acknowledgements This work was supported in part by a grant from the Silicon Valley Community Foundation (UBRI), grant no.2022-316136.

Author contributions Hjartarson-contribution as part of Ph.D. research, in particular on framework and architecture and bringing formalism to the work. Thrastarson-contribution as part of MS thesis work, in particular implementation and experimentation. Prof. Anna Sigriour Islind-co-advisor to Mr. Hjartarson on this work and member of the M.Sc. committee for Mr. Thrastarson. Prof. Gísli Hjálmtýsson, -primary advisor of Mr. Hjartarson, and of Mr. Thrastarson, director of the Fintech Center at Reykjavik University and creator of the Annáll blockchain.

Funding This work was supported in part by a grant from the Silicon Valley Community Foundation (UBRI), grant no.2022-316136.

Data availability Not applicable.

Declarations

Conflict of interest Not applicable.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. Quayson, M., Bai, C., Sun, L., Sarkis, J.: Building blockchain-driven dynamic capabilities for developing circular supply chain: Rethinking the role of sensing, seizing, and reconfiguring. *Bus. Strategy Environ.* **32**(7), 4821–4840 (2023)
2. Chase, B., MacBrough, E.: Analysis of the xrp ledger consensus protocol, 2018. [Online]. Available: <https://arxiv.org/abs/1802.07242>
3. Brown, R.G., Carlyle, J., Grigg, I., Hearn, M.: Corda: an introduction, R3 CEV, August, vol. 1, no. 15, p. 14, 2016
4. Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., Caro, A.D., Enyeart, D., Ferris, C., Laventman, G., Manevich, Y., et al.: Hyperledger fabric: a distributed operating system for permissioned blockchains. In: *Proceedings of the Thirteenth EuroSys Conference*, pp. 1–15 (2018)
5. Buterin, V., et al.: A next-generation smart contract and decentralized application platform, White Paper (2014)
6. Lindman, J., Tuunainen, V.K., Rossi, M.: Opportunities and risks of blockchain technologies—a research agenda, Hawaii International Conference on System Sciences (2017)

7. Xu, X., Sun, G., Luo, L., Cao, H., Yu, H., Vasilakos, A.V.: Latency performance modeling and analysis for hyperledger fabric blockchain network. *Inf. Process. Manag.* **58**(1), 102436 (2021)
8. Hyla, T., Pejaš, J.: ehealth integrity model based on permissioned blockchain. *Future Internet* **11**(3), 76 (2019)
9. Thantharate, P., Thantharate, A.: Zerotrustedblock: Enhancing security, privacy, and interoperability of sensitive data through zerotrusted permissioned blockchain. *Big Data Cognitive Comput.* **7**(4), 165 (2023)
10. Xiang, Z., Malkhi, D., Nayak, K., Ren, L.: Strengthened fault tolerance in byzantine fault tolerant replication. In: 2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS). IEEE, pp. 205–215 (2021)
11. Solat, S., Calvez, P., Nait-Abdesselam, F.: Permissioned vs. permissionless blockchain: How and why there is only one right choice. *J. Softw.* **16**(3), 95–106 (2021)
12. Liu, M., Wu, K., Xu, J.J.: How Will Blockchain Technology Impact Auditing and Accounting: Permissionless versus Permissioned Blockchain, *Current Issues in Auditing*, vol. 13, no. 2, pp. A19–A29 (2019). [Online]. Available: <https://doi.org/10.2308/cia-52540>
13. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008). [Online]. Available: <https://bitcoin.org/bitcoin.pdf>
14. Castro, E. A. M., Liskov, B.: Practical byzantine fault tolerance. In: *OSDI*, vol. 99, pp. 173–186 (1999)
15. Amores-Sesar, I., Cachin, C., Mičić, J.: Security analysis of ripple consensus (2020)
16. Lamport, L.: Paxos made simple, fast, and byzantine. In: *OPODIS*, ser. *Studia Informatica Universalis*, A. Bui and H. Fouchal, Eds., vol. 3. Suger, Saint-Denis, rue Catulienne, France, 2002, pp. 7–9. [Online]. Available: <http://dblp.uni-trier.de/db/conf/opodis/opodis02.html#Lamport02>
17. Ongaro, D., Ousterhout, J.: In search of an understandable consensus algorithm. In: *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*, ser. *USENIX ATC '14*. USA: USENIX Association, pp. 305–320 (2014)
18. Castro, M.: Practical byzantine fault tolerance (2001)
19. Amoussou-Guenou, Y., Pozzo, A.D., Tucci-Piergiorganni, S.: Tendermint. Springer, Cham, pp. 195–221 (2024). [Online]. Available: https://doi.org/10.1007/978-3-031-32146-7_7
20. Lu, Y., Lu, Z., Tang, Q.: Bolt-dumbo transformer: asynchronous consensus as fast as the pipelined bft. In: *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, ser. *CCS '22*. New York, NY, USA: Association for Computing Machinery, p. 2159–2173 (2022). [Online]. Available: <https://doi.org/10.1145/3548606.3559346>
21. Wang, H., You, Q., Duan, S.: Synchronous byzantine agreement with $o(n)$ messages and $o(1)$ expected time. *IEEE Trans. Inf. Forensics Secur.* **20**, 338–349 (2025)
22. Liu, A., Liu, Y., Wu, Q., Zhao, B., Li, D., Lu, Y., Lu, R., Susilo, W.: Cherubim: A secure and highly parallel cross-shard consensus using quadruple pipelined two-phase commit for sharding blockchains. *IEEE Trans. Inf. Forensics Secur.* **19**, 3178–3193 (2024)
23. Liu, Y., Xing, X., Cheng, H., Li, D., Guan, Z., Liu, J., Wu, Q.: A flexible sharding blockchain protocol based on cross-shard byzantine fault tolerance. *IEEE Trans. Inf. Forensics Secur.* **18**, 2276–2291 (2023)
24. Liu, Y., Liu, A., Lu, Y., Pan, Z., Li, Y., Liu, J., Bian, S., Conti, M.: Kronos: a secure and generic sharding blockchain consensus with optimized overhead. *Cryptology ePrint Archive*, Paper 2024/206 (2024). [Online]. Available: <https://eprint.iacr.org/2024/206>
25. Sayeed, S., Pitropakis, N., Buchanan, W., Markakis, E., Papat-saroucha, D., Politis, I.: Trustee: towards the creation of secure, trustworthy and privacy-preserving framework, pp. 1–10 (2023)
26. Kushwaha, S.S., Joshi, S., Singh, D., Kaur, M., Lee, H.-N.: Systematic review of security vulnerabilities in ethereum blockchain smart contract. *IEEE Access* **10**, 6605–6621 (2022)
27. Kim, C.: Ethereum 2.0: How it works and why it matters, Coindesk: <https://www.coindesk.com/wp-content/uploads/2020/07/ETH-2.0-072120.pdf> (2020)
28. Grandjean, D., Heimbach, L., Wattenhofer, R.: Ethereum proof-of-stake consensus layer: participation and decentralization (2023). [Online]. Available: <https://arxiv.org/abs/2306.10777>
29. Buchman, E.: Tendermint: Byzantine fault tolerance in the age of blockchains. Ph.D. dissertation, University of Guelph (2016)
30. Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., De Caro, A., Enyeart, D., Ferris, C., Laventman, G., Manevich, Y., Muralidharan, S., Murthy, C., Nguyen, B., Sethi, M., Singh, G., Smith, K., Sorniotti, A., Stathakopoulou, C., Vukolić, M., Cocco, S. W., Yellick, J.: Hyperledger fabric: a distributed operating system for permissioned blockchains, in *Proceedings of the Thirteenth EuroSys Conference*, ser. *EuroSys '18*. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: <https://doi.org/10.1145/3190508.3190538>
31. Gällersdörfer, U., Strugala, J.-N., Matthes, F.: Efficient onboarding and management of members in permissioned blockchain networks utilizing tls certificates. *Front. Blockchain*, **4**, 739431 (2021)
32. Ghosh, B.C., Ramakrishna, V., Govindarajan, C., Behl, D., Karunamoorthy, D., Abebe, E., Chakraborty, S.: Decentralized cross-network identity management for blockchain interoperability. In: *IEEE International Conference on Blockchain and Cryptocurrency (ICBC) 2021*, 1–9 (2021)
33. Psarra, E., Apostolou, D., Verginadis, Y., Patiniotakis, I., Mentzas, G.: Permissioned blockchain network for proactive access control to electronic health records, *BMC Medical Informatics and Decision Making*, vol. 24, no. 1, p. 303, 2024. [Online]. Available: <https://bmcmmedinformdecismak.biomedcentral.com/articles/10.1186/s12911-024-02708-8>
34. Hjalmtýsson, G., Sigurdsson, S.: Resource efficient peer-to-peer timestamping service [unpublished whitepaper] (2020)
35. Massias, H., Avila, X., Quisquater, J.-J.: Design of a secure timestamping service with minimal trust requirement (2002)
36. Merkle, R.: Protocols for public key cryptosystems **04**, 122–134 (1980)
37. Vernam, G.S.: Cipher printing telegraph systems for secret wire and radio telegraphic communications, *Transactions of the American Institute of Electrical Engineers*, vol. XLV, pp. 295–301, 1926
38. Alpern, B., Schneider, F.B.: Defining liveness, *Information Processing Letters*, vol. 21, no. 4, pp. 181–185, 1985. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/0020019085900560>

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Hafsteinn Hjartarson is a Ph.D. student at the Department of Computer Science at Reykjavík University. As a part of Reykjavík University's FinTech Center, Hafsteinn's main research areas are distributed computer systems in general and permissioned blockchains in particular. He holds an MSc in Software Engineering from Reykjavík University and has vast industry experience in Information Technology.



Fjölur Þrastarson holds an MSc in Computer Science with a focus on distributed systems and a BSc in Business Administration from Reykjavík University. Currently, he is a Software Engineer at Síminn Pay, where he has developed a B2B expense management system tailored for the Icelandic market. During his MSc studies, he was part of Reykjavík University's Fintech Centre, where he contributed to research on novel membership management solu-

tions, publishing an article on the topic.



Anna Sigridur Islind has a Ph.D. from University West in Sweden. She is an Associate Professor at the Department of Computer Science at Reykjavík University and the director of the research center CISDAS: Centre for Information Systems and Data Science Research. Moreover, she is the leader of the digital innovation in the Sleep Revolution project, a large-scale Horizon 2020 project with 39 partners across Europe funded by the European

Union, and she is the program director for the M.Sc. in Digital Health,

a novel multi-disciplinary study line at Reykjavík University. She has published well over 100 peer-reviewed papers, most of which focus on doing good through information systems by being socially aware and ethically driven during the design, development, and use of digital platforms, data and artificial intelligence.



Gísli Hjálmtýsson is a Professor of computer science whose research interests are in networking and distributed systems. He earned his B.Sc. in Applied Mathematics and Computer Science from the University of Rochester and completed his Ph.D. in Computer Science at the University of California, Santa Barbara in 1995. Following his doctorate, he worked as a researcher at AT&T Bell Laboratories at Murray Hills, and later ATT

Shannon Laboratories and held an adjunct professorship at Columbia University. He has played a key role in shaping computer science education in Iceland, serving as Dean of the School of Computer Science at Reykjavík University (RU) from 2001 to 2007 and returning to the position in 2017, and as Dean of School of Technology from 2019-2022. Dr. Hjálmtýsson has published over 80 peer-reviewed papers and holds more than 20 patents, with research interests in networking and distributed systems and services.