Resilient and Secure Distributed Ledgers: Adversary Models, Efficient Consensus, and System Design

HANS SCHMIEDEL

M.Sc

Supervisor: Jiangshan Yu
Associate Supervisor: Qiang Tang
Associate Supervisor: Ron Steinfeld

A thesis submitted in fulfilment of
the requirements for the degree of
Doctor of Philosophy

School of Computer Science
Faculty of Engineering
The University of Sydney
Australia

30 September 2025

Statement of originality

I hereby certify that, to the best of my knowledge, the content of this thesis is my own work and has not been submitted for any degree or other academic purposes.

I further declare that the intellectual content of this thesis is the result of my independent research, and all sources of assistance and information have been properly acknowledged.

Hans Schmiedel, June 2025

Abstract

Byzantine Fault-Tolerant (BFT) protocols enable distributed ledgers to operate without relying on a single trusted party, tolerating up to a threshold of Byzantine faults among participants. While traditional BFT was designed for closed systems with a small number of participants over local networks, blockchain applications have shifted the setting to open systems with thousands of participants connected over the internet. This transition introduces two critical vulnerabilities unaddressed by classical BFT models: (1) the susceptibility of nodes to internet Denial-of-Service (DoS) attacks, and (2) the potential for correlated failures when participants use similar configurations, violating the fundamental assumption of fault independence.

This thesis addresses these vulnerabilities through three main contributions. First, the Mobile Crash Adaptive Byzantine (MCAB) adversary model is introduced, which captures mobile DoS attacks by allowing the adversary to crash up to c nodes that can change over time, in addition to f Byzantine faults. Protocols are proven to require either *concealment* (hiding node identities until after broadcasting) or *abundance* (having more nodes perform each role than the adversary can target) to maintain liveness under MCAB. Using these results, it is shown that protocols designed for public blockchains like Algorand naturally achieve security against a stronger model than security proofs have previously indicated, the MCAB model. In contrast, protocols strictly designed for the traditional BFT setting with leader rotations like Hotstuff lose liveness.

Second, the modern Directed Acyclic Graph (DAG) based BFT line of work is expanded for system models captured by the MCAB model. The first constant latency dynamically available DAG based BFT protocol is proposed, achieving 3Δ expected latency where Δ is the network delay bound. To finalize subsets of a shared DAG among nodes, a novel primitive is introduced, *Graded Common Prefix* (GCP). GCP enables nodes to agree on a common subset of a DAG in just 2 communication steps, compared to 4 steps required by standard

consensus. By combining the dynamically available DAG protocol with graded common prefix, a flexible protocol is proposed. It allows clients to choose between prioritizing liveness or safety, while benefiting from the high performance of modern DAG BFT. In addition, it is shown using the MCAB model that this flexible protocol can relax the Global Awake Time assumption of previous flexible protocols with similar properties.

Third, this thesis addresses fault independence through incentive mechanisms that encourage nodes to adopt diverse configurations, such as running different node software implementations. It is proven that avoiding a single point of failure requires at least $\lceil n/f \rceil$ equally distributed configurations where f is the number of Byzantine faults and n the current total number of nodes, implying at least 4 concrete configuration versions when $f < n/3$. The challenge to incentivize diverse configurations is that costs related to the wide variety of configurations, from software implementation to geo-location, are hard to quantify. To circumvent this challenge, control mechanisms from reinforcement learning and control theory are leveraged, as they function without analytical solutions to the underlying system. Three adaptive reward allocation mechanisms are proposed and extensively evaluated: a Ziegler-Nichols tuned PID controller, a reinforcement learning tuned PID controller, and a pure reinforcement learning approach. Our simulations demonstrate that the traditional PID controller converges fastest and handles configuration additions most gracefully, while RL-based approaches better adapt to extreme cost variations.

Together, these contributions advance the resilience of BFT protocols for internet deployments by addressing both adversarial attacks and correlated failures. Formal models and theoretical foundations are provided, along with practical protocols for more robust distributed ledgers.

Acknowledgements

I would like to thank my supervisor Associate Professor Jiangshan Yu for guiding this thesis' research. Jiangshan Yu has provided me with invaluable advice and close mentorship on every aspect of my PhD. He has taught me about impactful research, academic writing, and how to navigate the academic world. Outside of research, I am grateful to be able to call Jiangshan Yu a reliable friend and mentor.

I would also like to thank my co-supervisors Associate Professor Ron Steinfeld and Associate Professor Qiang Tang. Ron Steinfeld has shared his extensive experience in cryptographic research, while supporting my academic journey with valuable feedback. Qiang Tang has provided me with deep insights in consensus research and constructive criticism that has greatly improved my work.

I would also like to thank my co-author Runchao Han, for keeping some of his valuable time to support my research. Runchao Han has regularly shared great feedback and suggestions that have always furthered my research.

My fellow PhD students have also been a great support throughout this journey, having welcomed me to the faculty and Australia with open arms.

I thank the Australian Research Council (ARC) for funding my research and making this journey possible.

Finally I would like to thank my friends and family for their constant encouragement and unwavering belief.

Author attribution

The content of this thesis is based on the following works. I took the lead in formulating research problems, proposing methods, conducting experiments, and drafting manuscripts in these works. Generative AI services were used to obtain writing suggestions and edit mathematical formulas.

Chapter 3 of this thesis has been *published* as: Hans Schmiedel, Runchao Han, Qiang Tang, Ron Steinfeld, and Jiangshan Yu. *"Modeling Mobile Crash in Byzantine Consensus."* In 2024 IEEE 37th Computer Security Foundations Symposium (CSF), pp. 159-171. IEEE, 2024.

Chapter 4 of this thesis has been *published* as : Hans Schmiedel, Runchao Han, Qiang Tang, Ron Steinfeld, and Jiangshan Yu, *"Constant Latency and Finality for Dynamically Available DAG"* in 2025 IEEE Symposium on Security and Privacy (SP), pp. 1910-1927, IEEE, 2025.

Chapter 5 of this thesis is being *prepared for submission* as: Hans Schmiedel, Runchao Han, Qiang Tang, Ron Steinfeld, and Jiangshan Yu, *"Incentivizing Fault Independence in Blockchains"* in 2026 Network and Distributed System Security (NDSS) Symposium.

In addition to the authorship attribution statements above, in cases where I am not the corresponding author of a published item, permission to include the published material has been granted by the corresponding author.

Hans Schmiedel, 20 June 2025

As supervisor for the candidature upon which this thesis is based, I can confirm that the authorship attribution statements above are correct.

Jiangshan Yu, 26 June 2025

Contents

List of Figures

CHAPTER 1

Introduction

1.1 Background and motivations

Distributed ledgers remove the reliance on a single party, for example a server or bank, to maintain a sequence of ledger entries. Deployed applications include resilient databases [**2**], electronic cash [**1**, **102**], and decentralized computing platforms [**3**]. A set of parties, called nodes, maintain the distributed ledger by reaching consensus on the sequence of added entries. To avoid relying on any single node, the distributed ledger must tolerate arbitrary misbehavior by nodes as much as possible. A misbehaving node, called Byzantine node, may represent a technical faults of a server, or node operators attempting to gain some personal benefit by compromising the ledger.

To obtain secure distributed ledgers, Byzantine Fault Tolerant (BFT) consensus protocols are therefore crucial. A BFT protocol's security is commonly defined by the threshold of Byzantine nodes it can tolerate, with respect to its current total number n of nodes. Traditionally, BFT has been designed for a low number ($n < 10$) of nodes. The performance and scalability demands of cryptocurrencies after the introduction of Bitcoin [**102**] have encouraged large advances in BFT protocols, solving the throughput and latency bottlenecks of consensus while greatly increasing the possible number of participants. Consensus throughput has increased from single digits to hundreds of thousands of added transactions per second [**46**], while latency to confirm a single transaction has been reduced from hours to under a second [**19**].

However, subtle differences between traditional BFT and blockchain protocols reveal research gaps addressed in this thesis. BFT originally considers local area networks involving just

a few nodes as in a datacenter or within an aircraft electronics. In contrast, the blockchain and cryptocurrency applications change the setting to involve a large number of unknown nodes connected over the internet. As a result, two major vulnerabilities are introduced by the transition to a globally distributed, open system:

- The network availability of nodes is vulnerable to internet Denial-of-Service (DoS) attacks.
- The server configuration of nodes can not be enforced, and many participants (more than f) with similar configurations, such as the same software libraries, may fail simultaneously. Tolerating f faults in theory may mean tolerating only one fault, for example a software bug, in practice.

DoS attacks on cryptocurrencies have already been attempted with some success [**76**, **90**], while multiple cryptocurrencies have halted due to a single software bug that affected all nodes [**57**, **87**].

This thesis addresses these vulnerabilities and accordingly further expands the space of modern BFT for internet deployments.

1.2 Research questions and results.

1.2.1 Modeling internet DoS attacks.

This thesis first studies the unaddressed problem of Denial-of-Service (DoS) attacks in an open internet system such as public blockchains.

In blockchain systems and BFT, the network is commonly modeled as either a *synchronous network*, where messages are assumed to deliver within a known bound, an *asynchronous network*, where messages deliver within an unknown bound, or a *partially synchronous network*, where the network first behaves as an asynchronous network then as a synchronous network after an unknown *Global Stabilization Time* (GST).

Many traditional BFT protocols use leaders to propose the next value, guaranteeing security against an adaptive Byzantine adversary who corrupts up to f nodes in a synchronous or partially synchronous network to avoid the FLP impossibility in asynchronous networks [**56**]. Leaders can be rotated if no value is received after a timeout, assuming that under the synchronous network (or after GST in partially synchronous networks), honest leaders' messages are delivered before a known delay.

On the internet, a simple DoS attack on leaders can fully stall classical leader-based BFT where leaders are known a priori and can be targeted at each timeout. This simple attack is not covered by classic security models of BFT, and protocols therefore do not tackle such an attack, or only provide informal discussions. Liveness proofs for BFT protocols such as PBFT [**36**] and Hotstuff [**130**] rely on the adversary only controlling up to f nodes. Within that model, leaders can only be successively faulty for f rounds before honest and online replicas are guaranteed to be chosen. This guarantee is only an assumption built in to the system model's definition. In practice, mounting a simple DoS attack that keeps targeting each new leader doesn't require large resources.

This gap leads to the first research question:

Research Question 1 (RQ1): *Can we design a generic system model that captures the public internet setting of a blockchain and identify the necessary properties protocols must achieve for security and liveness ?*

RQ1 has been answered in the affirmative in Chapter 3. The Mobile Crash Adaptive Byzantine (MCAB) adversary is introduced. The adversary has access to f Byzantine faults selected adaptively, which means as the protocol runs and Byzantine nodes remain Byzantine for the duration of the protocol. In addition, the adversary has access to c mobile crash faults at any time during the protocol, which means up to c nodes are crashed but the set of crashed nodes may be different as the protocol runs.

The widely used adversary with up to f faults [**9**, **29**, **35**, **36**, **48**, **61**, **74**, **89**, **97**, **130**], selected either at the onset of the protocol (static) or during (adaptive), implicitly assumes $c = 0$ and ignores possible DoS attacks on more than f successive nodes.

In contrast, various models allow mobile crashes [**9**, **72**, **75**, **83**, **100**, **107**] but without any parameter definition, implying that c is unbounded. As a result, these models operate in the dynamic availability setting, where nodes join and leave at any time, and are subject to the CAP impossibility theorem for partition tolerant protocols [**68**].

The MCAB model, with its flexible parameters f and c, provides a generic framework encompassing a wide range of desired system properties for internet deployments, illustrated in Figure 3.1 and summarized in Table 3.1 in Chapter 3.

Two protocol properties are defined, *Concealment* and *Abundance*, and it is shown that either is necessary and sufficient to maintain liveness in the MCAB model. Concealment formally defines when a protocol's next broadcasting node identity is hidden until it the broadcast is complete. Abundance formally defines when a large number of nodes perform a similar broadcast such that the adversary can not crash all of them simultaneously (parameterized by c). The performance tradeoffs of each design principle are explored, deriving bounds on communication rounds and message complexity for each. As case studies, we study a few example protocols.

It is shown that Algorand [**39**, **67**], a protocol formally designed for the adaptive Byzantine adversary, but informally meant to tolerate DoS attacks, is actually secure against the stronger MCAB adversary. Hotstuff [**130**], a protocol stricly designed for an adaptive Byzantine adversary (assuming adaptively secure cryptography), is, as expected, shown to lose liveness against the MCAB adversary as leaders are vulnerable to DoS attacks.

1.2.2 DAG-based BFT for flexible protocols.

The second research gap explored in this thesis regards Directed Acyclic Graph (DAG) based consensus for dynamic availability and finality.

Protocols with dynamic availability are desired in practice, exemplified by the usage of dynamically available consensus protocols in both of the largest cryptocurrencies by market cap, Bitcoin [**61**] and Ethereum [**33**].

In Ethereum's case, another protocol runs in parallel to provide partition tolerant finality to the ledger, an approach formalized in the Ebb-and-flow framework [**104**]. Running two consensus sub-protcols in parallel allows a distributed ledger to circumvent the CAP theorem [**68**], providing both dynamic availability and partition tolerance. However, two distinct system models are involved, requiring the arguably strong assumption that all honest nodes are eventually online, at a Global Awake Time (GAT). Following the results from **RQ1**, this naturally leads to the following research question:

RQ2: *Can we weaken the GAT assumption for flexible protocols that concurrently run a dynamically available sub-protocol and a partition tolerant sub-protocol?*

RQ2 is answered in Chapter 4. Leveraging the MCAB definition, a weaker version of GAT, Quorum Awake Time (QAT) is defined where *enough* honest nodes are eventually awake. The specific number that satisfies *enough* depends on the finality protocol in question, and can be parameterized with the parameter c of the MCAB model.

Both Bitcoin and Ethereum run dynamically available blockchain protocols to obtain a distributed ledger. As such, they do not benefit from the modern advances in DAG-based protocols. DAG-based protocols build a DAG of blocks, as opposed to a single blockchain. To do so, participants intentionally propose blocks simultaneously, improving throughput over the blockhain approach where a single participant proposes a block. In addition, DAG protcols tend to be well suited for the MCAB model proposed for **RQ1**. Parallel proposals naturally provide abundance, preventing the simple DoS attacks on single block proposers.

However, DAG BFT protocols have focused on partition tolerant protocols, and no DAG BFT protocol with constant expected latency previously existed with dynamic availability (unbounded parameter c in the MCAB model). A natural research question is therefore to ask:

RQ3: *Can we design a DAG-based BFT protocol with constant expected latency in the dynamic availability setting?*

RQ3 is answered in Chapter 4 by the introduction of a DAG BFT protocol with constant latency and dynamic availability. Evaluations show that the throughput scales with the number

of nodes, unlike existing constant latency dynamically available protocols that do not benefit from the DAG structure.

For the finalization process, Ethereum utilizes a protocol specific finality gadget [**33**] without formal security proofs. Attacks have been found [**113**] and the Ebb-and-Flow framework has been proposed [**104**] to achieve Ethereum's desired properties, dynamic availability and finality. The proposed finality sub-protocol is a generic BFT protocols such as Hotstuff [**130**], operating independently from the dynamically available blockchain.

In the DAG BFT line of work, DAG finalization is done using DAG-specific mechanisms using leaders [**121**] or common coins [**46**]. These mechanism are not generic BFT protocols, and leverage the DAG structure to finalize the DAG. This gap between the Ebb-and-Flow finality protocol and modern DAG BFT leads to the following research question:

RQ4: *Can we use the information contained in a blockchain or DAG to obtain deterministic agreement without using full consensus?*

RQ4 is answered in Chapter 4 by the introduction of a new primitive, *graded common prefix*, that allows a set of nodes to agree on the subset of a DAG. The definition of DAG is formalized as *block-DAG*, where a single blockchain is a special case. A construction of *graded common prefix* is shown to achieve lower latency than using full consensus such as Hotstuff to finalize a block-DAG.

1.2.3 Fault independence for public distributed ledgers.

The third research gap addresses the potential for correlated faults in a public distributed ledger. The previous research areas of this thesis have addressed the adversary model for the internet, and expanded the knowledge of modern DAG BFT in that context. However, in deployed systems, a single fault common to most nodes can immediately corrupt more nodes than tolerated by any high performance protocol and respective adversary models. Such system failures have happened in recent years in multiple cryptocurrencies [**57**, **76**, **87**].

The key to this vulnerability is the implicit assumption that faulty nodes of a BFT protocol are independently faulty. Ideally, each participant in a BFT protocol is completely independent, and up to f distinct faults of any kind are tolerated. In traditional distributed settings, this can be manually enforced However in practice, multiple participants may be using the same hardware, software, network infrastructure, datacenter provider, and any other relevant server configuration aspect.

This gaps leads to the last research question of this thesis: **RQ5**: *Can fault independence in an open system be reliably achieved through incentive mechanisms?*

RQ5 is answered by the proposal of three incentive mechanisms that encourage diverse configurations. These mechanisms are agnostic to the considered configuration aspect such as software implementation or geo-location. A simulation framework was developed to model rational node behavior in different incentive mechanisms, showing in which conditions each proposed mechanism is able to achieve ideal fault independence.

1.3 Thesis outline

This thesis is organized as follows:

Chapter 2 reviews the relevant literature on consensus models, DAG-based protocols, and fault independence in distributed systems.

Chapter 3 addresses **RQ1** by introducing the Mobile Crash Adaptive Byzantine (MCAB) adversary model to capture mobile DoS attacks in internet deployments. The chapter defines the properties of Concealment and Abundance as necessary and sufficient conditions for liveness, and analyzes existing protocols under this model.

Chapter 4 addresses **RQ2**, **RQ3**, and **RQ4** through three main contributions. Section 4.5.1 addresses **RQ2** by generalizing the Global Awake Time assumption to Quorum Awake Time using the MCAB model for more realistic participation assumptions in flexible protocols. Section 4.3 addresses **RQ3** by presenting the first constant latency dynamically available

DAG protocol achieving 3Δ expected latency with throughput that scales with the number of nodes. Section 4.4 addresses **RQ4** by introducing Graded Common Prefix as an efficient primitive for deterministic agreement on DAG subsets without requiring full consensus.

Chapter 5 addresses **RQ5** by proposing three incentive mechanisms to achieve fault independence in open systems. The chapter establishes theoretical requirements for avoiding single points of failure and evaluates the proposed mechanisms through simulation.

Chapter 6 concludes with a summary of contributions and directions for future work.

CHAPTER 2

Literature review

The literature in blockchain stems from both key whitepapers like Bitcoin and traditional publications in distributed computing and cryptography.

This chapter is divided into the following sections: Section 2.1 presents consensus work. Section 2.2 compiles the existing forms of mobile adversaries. Section 2.3 reviews existing work on fault independence and remote attestation mechanisms. Finally, section 2.4 discusses the additional cryptographic primitives the blockchain setting might need.

2.1 Consensus for blockchains

2.1.1 Nakamoto consensus

Nakamoto consensus, also called longest chain protocols, was introduced by Nakamoto [**102**] and expanded into different variations such as GHOST [**119**] and Ouroboros [**48**]. Their major difference to classical BFT protocols is that they offer probabilistic safety and guaranteed liveness, as opposed to guaranteed safety and probabilistic liveness (when augmented with randomness). At a high level, Nakamoto consensus works by having participants randomly selected to propose a block, built on top of their local view of the longest chain. By assuming that $> 50\%$ of selected participants are honest, a block can be guaranteed to be part of the canonical chain once enough newer blocks are built on top. Formal analysis was done by Garay et al. [**61**] and Dembo et al. [**50**]. The random selection can be done in various ways, most notably PoW as in Bitcoin [**102**] and PoS using weighted VRF outputs as in Ouroboros Praos and its follow up works [**48**] [**20**] [**21**].

At first glance, Nakamoto consensus seems well suited to handle mobile adversaries since the winners of the random selection for new blocks are anonymous until they publish their block, at which point adaptively corrupting them is not particularly useful. However, the formal models of the adversary to analyze their security are defined differently than BFT protocols due to their permissionless nature where the number of participants and their share of the hash power or stake is in constant flux. The adversary therefore controls hash power or stake as opposed to a number of nodes.

2.1.1.1 Bitcoin

Garay et al. [**61**] formalize the properties for a robust transaction ledger through persistence and liveness, and show that the common prefix and chain quality properties of a blockchain are enough to fulfill them. Garay et al. and call the adversary adaptive since its corruptions aren't fixed at the start of the protocol, but contrary to BFT adaptive adversaries, isn't limited to only f corruptions for the whole execution. The assumption for this adversary is that it never controls more than 50% of total hash power. similarly to the mobile adversaries of existing PSS and BFT work. Since the Bitcoin protocol doesn't have specific executions defined as for BFT protocols, an adaptive adversary for Bitcoin behaves like a mobile adversary in BFT consensus.

2.1.1.2 Ouroboros

Ouroboros Praos [**48**] leverages proof-of-stake weighted VRF to replace Bitcoin's PoW lottery. Participants with winning VRF outputs are the equivalent of a miner who found a valid hash output. Additional considerations must be made for a common seed input to the VRF, which Ouroboros Praos adds in the chain and analyzes an adversary's capacity to bias this beacon. Security is proven against an adaptive adversary bounded in the fraction of stake it controls for the execution of a single epoch, a predefined amount of time. It is then lifted to continuous executions of epochs including stake distribution shifts, implying a mobile adversary that adaptively corrupts up to the defined fraction of stake each epoch. Ouroboros Genesis [**21**] further builds on Ouroboros' foundations to prove its security properties in the Universally

Composable model and provides a specifications for new parties to securely join a running protocol.

2.1.1.3 Everything is a race and Nakamoto always wins

Dembo et al. [**50**] compare the security guarantees of PoS and PoW and conclude that they are ultimately equivalent when considering the adversary's capacity to overtake honest forks. This is shown in spite of the fact that PoS longest chain protocols enable new adversarial behaviors such as nothing-at-stake attacks, impossible in PoW.

2.1.2 BFT protocols

Modern permissioned BFT stems from PBFT [**36**] and its leader based consensus with quorum certificates. For the blockchain setting where frequent leader rotations are desirable, the view change has been optimized to be executed every block in Tendermint [**29**] and HotStuff [**130**]. By using threshold signatures to form quorum certificates, Hotstuff [**130**] achieves BFT in linear communication complexity for the first time and optimistic responsiveness, while Hotstuff 2 [**92**] further improves the protocol by one round. Optimistic responsiveness lets the protocol advance at the pace of the network, without having to wait for a full network delay between rounds if not necessary. The chained HotStuff version still needs three consecutive leaders to commit a block, an issue addressed by Jalazai et al. [**80**] and in Marlin [**124**]. By assuming an f bounded adversary, liveness can be achieved in the worst case where the first f leaders are faulty, and the leader $f + 1$ would have to behave honestly.

2.1.2.1 Quorum certificate BFT

In BFT consisting of $n > 3f + 1$ replicas, quorum certificates are formed by $2f + 1$ cryptographic signatures from different replicas on the same content. BFT protocols leverage this tool to confirm blocks proposed by a leader in the following way: A leader proposes a block b and sends it to all replicas, honest replicas echo their signature on b to enable a

quorum certificate once $2f + 1$ signatures are collected. With the help of network assumptions and sequential quorum certificates on b, protocols such as PBFT [**36**], Tendermint [**29**], Hotstuff [**130**], ensure the safety of committed blocks, as conflicting blocks require quorum certificates and therefore the contribution of $f + 1$ honest replicas. Quorum certificates can be implemented by multiple single signatures or by aggregating them with threshold cryptography. We assume either adaptive secure threshold signatures or single signatures to be able to consider adaptive adversaries for quorum certificate BFT.

2.1.2.2 Asynchronous BFT

Asynchronous BFT was made practical with Honeybadger [**97**], further optimized with the Dumbo [**89**], and made adaptive secure with EPIC [**86**]. In asynchronous BFT, the impossibility of differentiating faulty leaders from network delays creates the need for alternative approaches such as agreeing on a common subset of proposals with multi-valued-byzantine-agreement [**89**] or running parallel instances of asynchronous byzantine binary agreement [**97**]. Although more costly in communication complexity and latency, these methods also solve the problem of targeted DoS attacks, as no single replica is a leader crucial for advancing the protocol.

2.1.2.3 DAG based BFT.

Prism [**22**] and Narwhal [**46**] have demonstrated that throughput up to network capacity is possible by separating transaction data from consensus, for dynamically available and deterministically safe consensus respectively. The transaction data is structured into directed acyclic graphs (DAG), confirmed by a consensus mechanism, allowing the transaction layer to be scaled arbitrarily according to system specifications.

Advances in DAG based BFT have further increased throughput to hundreds of thousands of transactions per second, and reduced latency up to less than a second [**19**, **94**, **116**, **121**].

2.1.3 Algorand and YOSO

2.1.3.1 Algorand

Algorand [**67**] is a BFT protocol with full finality but is also defined in terms of the adversary's controlled stake in a permissionless setting. Algorand also selects nodes with VRF to not only propose blocks, but participate in BA voting rounds, ultimately achieving guaranteed consensus on accepted blocks. As the selection of block proposer and each BA voting round are independent, Algorand also informally provides strong security against mobile adversaries that would be unable to meaningfully target any node in particular. The authors' only assumption on the adversary is that it controls less than a fraction of stake, and informally mention that "Algorand is not succeptible to either targeted compromises or DoS attacks", implying the inclusion of mobile adversaries. However VRF's reliance on secret keys means adversaries can't be fully byzantine and mobile without an additional key refresh mechanism.

2.1.3.2 You Only Speak Once

Bitcoin 's and Algorand's way of randomly selecting parties for each protocol step was analyzed and formalized in You Only Speak Once (YOSO) [**66**]. However, although the authors mention the same DoS and gradual corruption of parties through zero-days as an example, this type of adversary that YOSO style protocols secure against is not explicitly defined. Rather, it is included in their broad "adaptive adversary" definition.

YOSO provides a framework for MPC where nodes are randomly selected to perform a role such that their identity is secret until the task is already complete. This is especially useful against adversaries adaptively corrupting the chosen nodes if their identity was public in advance, such as mobile adversaries. The authors also informally describe how the YOSO approach denies targeted DoS attacks. Gentry et al. [**66**] and Benhamouda et al. [**24**] provide ways to transfer the current state of computation between these anonymous nodes, which is actually unnecessary for our BFT applications where no secret states must be transferred between computation steps, as is required for MPC.

Consensus protocols like Bitcoin [**102**], Ouroboros Praos [**48**] and Algorand [**67**] fall under the YOSO umbrella as their block proposers (and voters in Algorand) are unknown in advance and only reveal themselves when broadcasting their input.

YOSO methods for randomly selecting anonymous nodes are directly applicable to our setting where leader based BFT can be improved against mobile and adaptive adversaries. Verifiable Random Functions (VRF) introduced by Micali et al. [**96**] and part of Ouroboros [**48**] and Algorand [**67**] are one way of selecting a leader anonymously by choosing the participant with the lowest VRF output under a certain threshold in a given time slot. The drawback is the need for synchronous communication with a known upper bound to avoid the consensus issues that arise with large delays in receiving the lowest VRF output.

2.1.4 Ebb-and-flow

Ebb-and-flow [**104**] proposes the first formal description of Ethereum 2.0's design goals: a dynamically available blockchain finalised by a deterministically safe protocol. This allows the overall protocol to maintain safety in periods of asynchronous network conditions, unlike standard longest chain protocols, while also maintaining liveness when participation is low. In addition, attacks against the Ethereum 2.0 candidate protocol [**33**] are presented, highlighting the need for formal security analysis.

2.1.5 Impossibility results

The seminal FLP result [**56**] shows that consensus with a single crash fault is impossible in an asynchronous network, without introducing randomness or network assumptions. Aguilera and Toueg show that consensus requires at least f+1 rounds of communication for f crash faults [**11**]. The result intuitively makes sense for leader based BFT, but crucially also applies to leaderless protocols without single points of failure. It has also been shown that agreement is impossible in a partially synchronous network if $f \geq n/3$ for f Byzantine nodes and n total nodes [**55**].

2.2 Mobile adversaries in distributed computing

The need for a mobile adversary model was first identified by Ostrovosky and Yung [**106**] to deal with malicious viruses for general multi party computation. The mobile adversary can control up to f participants at any point in time. This is defined through *pebbles* representing the corruptions that the adversary is free to place at any f processors. The entire state of pebbled processors is under the adversary's control and when a pebble is removed, processors go in a pre-defined reboot status and obtain a fresh state. The authors note that this definition also gives the adversary control over which processors recover, for example that a processor with a permanent pebble would stay corrupted forever.

"For any $\epsilon < 1$, *mobile ϵ-adversary* is an infinitely powerful machine with $t = \epsilon n$ pebbles which operate on an n processor network" [**106**]

Follow-up work on the mobile adversary refine the model depending on the intended context. We present the different versions in the following sections

2.2.1 Definitions of the mobile adversary

2.2.1.1 The window of vulnerability mobile adversary

Dividing time in different time windows helps bound the mobile adversary through a particular protocol's properties. As opposed to moving the pebbles every round, the adversary is limited by only moving pebbles every time window. This model was introduced by Herzberg et al. [**77**] and used by Castro and Liskov [**36**] to implement the first BFT SMR protocol in a mobile adversary setting. To achieve this property regularly executed reboot recovery mechanisms are proposed by the authors.

2.2.1.2 Dynamic committee mobile adversary

Dynamic committees were informally introduced by Zhou et al. [**133**] and defined by Schultz and Liskov [**112**] in terms of *epochs*. Each epoch the adversary can adaptively choose up to f

nodes at any point. At the end of the epoch, the nodes complete the handoff protocol to the next epoch's members. In this definition corrupted nodes remain corrupted forever, due to the adversary having had access to their secret keys. The authors argue that refreshing their keys for a future epoch is effectively the same as new nodes in the committee.

This definition is subsequently used for other dynamic committee proactive secret sharing works.

In this model, the total set of nodes changes over time, with a growing number of permanently corrupted nodes. By assuming the f corruption bound on each participating committe for their respective epoch, the authors avoid the need to specify recovery mechanisms and implicitly assume newly available honest nodes in each epoch.

"The adversary is assumed to be active and adaptive: it may decide to corrupt nodes at any point. We assume that the adversary can corrupt no more than a threshold f of the nodes in a group holding the secret in epoch e before the end of that epoch. Additional nodes may be corrupted after they have left the epoch" [**112**]

The specific membership mechanisms for different committees is abstracted and considered orthogonal work, such as the dynamic membership for BFT protocols, as formalized by Duan and Haibing [**54**].

2.2.1.3 Mobile fault model

Garay [**63**] defines the *Mobile Fault* model, where ρ defines the number of rounds it takes for a fault to corrupt a new processor, and leave the old one to be cured.

"the Mobile Fault model is the family of models defined by the tuple $(\frac{f}{n-1}, \rho)$ where the number of mobile faults is f, the total number of processors is n, and ρ is the roaming pace of the faults" [**63**]

Garay [**63**] further defines the Mobile Fault Byzantine Agreement (MBA) problem as an extension of BA with the added property of *Consistency Maintenance*

Consistency Maintenance: Once Agreement is reached amongst the current set of uninfected processors, it is preserved amongst the (possibly different) uninfected processors.

For $\rho = 1$, Garay [**63**] shows an impossibility result for deterministic MBA with $f \geq 1$. This can be circumvented by assuming 1 incorruptible processor during an execution or by adding randomness.

Buhrmann [**30**] uses Garay's MBA model but sets the new corruptions to be executed during the sending phase of a round, emulating viral propagation. This ensures the uncorrupted processors, including the cured ones, are fully functioning at all times. This weakens the adversary as opposed to Garay's model where, by moving corruptions just before the send phase, newly cured processors would have no information to contribute in the send phase, effectively acting as crashed for one extra round.

Sasaki [**110**] modify Garay's MBA setting to consider an even stronger mobile adversary setting where honest nodes don't know if they were just recovered, and the adversary fills their sending buffer with extra messages before they recover. This essentially means the adversary has up to $2f$ corruptions in some rounds, naturally limiting the resiliency to $n > 6f$.

Bonnet [**28**] chooses a compromise between Sasaki and Garay's MBA model by letting the adversary fill cured processors' buffer with corrupt messages but they still behave honestly, and can't send different messages to different processors for example.

2.2.2 Different context for mobile adversaries.

2.2.2.1 Byzantine fault tolerance

Castro [**36**] and Zhou et al. [**132**] use a mobile adversary defined by its window of vulnerability, whose length can be determined by the capabilities of the BFT protocol and the security requirements of the application.

Their work proposes specific rebooting mechanisms that integrate in the main protocol to proactively recover replicas. Proactive recovery is performed with a recovery protocol each

window of vulnerability for each node, regularly removing the adversary from corrupted nodes. Through the proactive recovery the protocols provide a mechanism to ensure the bounded assumptions on the mobile adversary, as opposed to abstracting how corrupted nodes recover as is the case in the byzantine agreement 2.2.1.3 line of work

Castro [**36**] introduces a mobile adversary model limited by the timeframe T in which it can corrupt f nodes. This timeframe is then defined long enough to accommodate a whole worst case execution of the PBFT protocol, including the sequential scheduled reboot recoveries of the nodes. The advantage of this approach is that the adversary acts like an adaptive adversary inside a timeframe, enabling existing protocols like PBFT without much modification to become secure for a mobile adversary.

The choice of a timeframe naturally introduces the synchrony assumption to bound the time it takes to run the required number of rounds. This could be avoided by defining the adversary's corruption budget to be withing a certain number of rounds, enough for the considered protocol to terminate.

2.2.2.2 Mobile adversaries and Byzantine agreement

The mobile adversary has also been introduced in byzantine a agreement by Garay [**63**] where the proactive recovery has been abstracted. The adversary is defined by its corruption speed ρ and the maximum number of corruptions f at any point during the protocol execution. Garay presents an impossibility result for determinstic byzantine agreement against a $\rho = 1$ adversary and shows the need for an incorruptible node assumption to obtain byzantine agreement against this adversary.

Buhrmann [**30**] shows the first optimal resiliency of $f \leq \frac{n}{3}$ protocol against a $\rho = 1$ mobile adversary. However, Buhrmann's mobile adversary moves exactly at the send phase each round, leaving the previous controlled nodes fully recovered. This contrasts to the Garay [**63**] and Banu [**23**] model where the adversary moves at any time during a round, potentially preventing the previous node from participating in the protocol for one extra round while it recovers, without counting in the adversary's corruption budget.

TABLE 2.1. Summary of existing work on the Mobile Fault Byzantine Agreement Problem.

Reference	adversary model	f	tight bound
Garay 1994	additional f can be crashed	$\frac{n}{6}$	no
Burhmann 1995	$2f + 1$ fully correct nodes	$\frac{n}{3}$	yes
Banu 2011	additional f can be crashed	$\frac{n}{4}$	open
Bonnet 2014	f extra corrupt buffer but not behavior	$\frac{n}{5}$	yes
Sasaki 2013	f additional corrupted behavior	$\frac{n}{6}$	yes

Banu et al. achieve $f < \frac{n}{4}$ resiliency in this model, improving over Garay's $f < \frac{n}{6}$ protocol, and conjecture its optimality under this model where f nodes are byzantine and f other nodes can be crashed at any time.

Sasaki et al. [**110**] further argue that honest nodes' knowledge of their recovery after a corruption shouldn't be given, and that their message buffer might still be filled with adversarial messages upon recovery. This gives the adversary control of up to $2f$ nodes' messages and the authors present a byzantine agreement protocol with a tight resiliency bound $f < \frac{n}{6}$. The bound being halved intuitively makes sense as the adversary can control twice as many nodes.

Bonnet et al. [**28**] define a model similar to Sasaki et al. [**110**] but limits the adversary's control over a cured nodes' sending buffer to its content but not the execution of the communication. For a concrete example, the adversary wouldn't be able to make a cured node send different messages in a broadcast, as is the case in the Sasaki model. Under such a model Bonnet et al. show a protocol with a tight bound of $t < \frac{n}{5}$

2.2.2.3 Asynchrony and the wormhole solution

When considering asynchronous networks, the adversary can always create an indefinite network delay while regularly corrupting nodes and surpassing f nodes before letting the network resume. The adversary must therefore also be limited by corruptions per round and not time units where the adversary could just take over the entire network with a large enough network delay.

Sousa et al. [**120**] propose a synchronous subsystem called wormhole with trusted components isolated from the rest of the network. This enables regular proactive recoveries even in an asynchronous network where preventing the adversary from corrupting the entire network would otherwise be impossible.

2.2.2.4 Proactive secret sharing

The Proactive Secret Sharing (PSS) line of work was introduced by Ostrovsky and Yung [**106**] as a theoretic way to enable MPC against a mobile adversary by regularly refreshing a secret's shared polynomials. Herzberg [**77**] adds computational assumptions on the adversary and a synchronous broadcast channel to use verifiable secret sharing and implement proactive secret sharing at regular time slot intervals. The considered mobile adversary is bounded with $f < \frac{n}{2}$ at any time slot.

Fully asynchronous PSS is introduced by Cachin et al. [**35**] while Zhou et al. [**133**] further add dynamic committees, the ability for the participating nodes to change between refreshes. Dynamic committees have the additional challenge that the adversary may control t nodes in the old and new committee, amounting to $2f$ shares of a threshold t shared secret. To solve this problem, Schultz et al. [**112**] assign an extra random polynomial sharing for each new member, reaching up to $O(n^4)$ communication complexity.

In asynchronous PSS, nodes must agree on the set of inputs making up the refreshed shares of the secret, creating the need for some kind of consensus protocol after the main resharing operations and adding extra round and communication complexity. Schultz et al. [**112**] therefore use PBFT's [**36**] advances in consensus to agree on the refreshed shares of the secret, introducing PBFT's own need for partial synchrony but achieving $O(n^4)$ message complexity. Shanrang [**128**] makes use of CHURP's [**95**] new bivariate polynomial resharing without the blockchain assumption for an asymptotic communication complexity gain to $O(n^3 log n)$. Rambaud and Urban [**108**] avoid the extra rounds necessary to obtain a consensus on the new shares and keep track of multiple valid resharings, knowing that at least one of them is able to reconstruct the secret. The asymptotic communication complexity is still $O(n^4)$ due to the large polynomials shared.

Another recent workaround for the broadcast channel is making use of a blockchain assumption as presented by Maram et al.'s CHURP [**95**], enabling more communication efficient synchronous PSS schemes. CHURP uses a new bivariate polynomials resharing scheme posted on the blockchain to perform dynamic membership PSS at only $O(n^2)$ communication cost but incurring $O(n)$ on chain cost, which may have other costs like fees depending on the blockchain. Goyal et al. [**71**] further amortize the communication cost per secret to $O(n)$.

Benhamouda et al. further expand the blockchain PSS approach by describing a way to anonymously select a sub committe to hold the secret and perform the resharing, letting the communication complexity scale with the size of the committee instead of the total number of nodes.

Efficient PSS protocols are necessary to deal with a mobile adversary for protocols using any kind of shared secret like threshold signatures. The faster a PSS protocol can refresh a secret, the faster the adversary model is allowed to move corruptions. PSS protocols optimized for quick termination are therefore a key building block for mobile adversary BFT.

2.2.3 Alternative network models

2.2.3.1 Sleepy model

The sleepy model of Pass and Shi [**107**] represents dynamic participation in a setting where not all honest nodes are willing to be online at all times, as opposed to an adversary aiming to prevent liveness of a fixed participation protocol in our model. An adaptive adversary in the sleepy model can adaptively choose which nodes sleep, becoming offline, and adaptively choose which nodes to corrupt, provided the protocol's bound on the honest fraction amongst the set of online nodes is respected.

A key difference between the sleepy model and our model is that our model can be applied to a partial synchrony setting. In synchrony, our model is weaker by considering the mobile crash faults as part of the total adversarial budget, instead of modelling dynamic participation. Algorand was not designed for the sleepy model and therefore does not guarantee liveness for

TABLE 2.2. Summary of works with slow mobile adversaries, their assumptions, contribution focus, and applicaton

Reference	assumptions	focus	application
PBFT 2002	trusted hardware	reboot mechanism with key renewal	BFT
Zhou 2002	fair links, administrative key pairs	reboot mechanism	distributed CA
Sousa 2010	wormhole crash-fault subsystem	proactive and reactive reboot	Intrusion-tolerant firewall
Cannetti and Herzberg 1994	authenticated point-to-point channels	privacy against eavesdropper	pseudo- randomness
Herzberg et al. 1994	broadcast channel	resharing protocol	PSS
CHURP 2019	blockchain, authenticated point-to-point	communication complexity	DPSS
Schultz 2010	PKI, authenticated point-to-point	communication complexity	DPSS
Benhamouda 2020	blockchain, PKI	evolving committee	DPSS
Baron et al. 2015	authenticated point-to-point	batching for $O(1)$ com. compl.	DPSS
Rambaud and Urban 2022	PKI, authenticated point-to-point	avoiding agreement	DPSS

it as the the fraction of honest and online users h can decrease, with the byzantine fraction decreasing accordingly, until the conditions are unlikely to be fulfilled. For example, if half of total users are offline, and 80% of online nodes are honest as in [**67**], we have a total honest fraction of 0.4 and the probability of liveness being violated becomes ≈ 1.

2.2.3.2 χ-weak-synchrony

In the χ-weakly-synchronous assumption, a fraction χ of all nodes are honest and online at any time and the set of honest and online nodes can vary each round.

The authors of [**75**] look to tolerate more than $1/3$ byzantine faults in this model, arguing that asynchronous protocols can be selected if considering a $1/3$ bound anyway. In contrast, our model tries to minimally modify the standard BFT model to include the additional adversarial capabilities protocols such as Algorand were designed for, and to enable high performance

protocols like Hotstuff to be lifted to a setting that also includes targeted DoS, without requiring large changes.

The χ-weakly-synchronous assumption inherently includes synchrony too, while our model can be applied to extend a partially synchronous setting. However the mobility of the χ-weakly-synchronous set of online honest nodes implies similar challenges where an adversary could adaptively set the online honest nodes to exclude any elected leaders.

2.2.3.3 *synchronous-k* model

Antoniadis et al. [**15**] tackle the same problem of vulnerable leaders by defining the *synchronous-k* model. Defined as a model where up to k replicas are suspended each round. The difference with our model is that such a model can prevent any leader based protocol from proceeding correctly, even if the leaders are unknown in advance. This motivates the authors' work on leaderless solutions for consensus.

2.3 Fault independence

Fault independence has been recently discussed by Yu [**131**], highlighting the need for replica diversity to achieve resilient systems. Replica (or node) diversity is measured using the Shannon Entropy, where the node distribution per configuration represents the probability distribution.

N-version programming has been introduced in the context of open blockchains, simultaneously executing multiple implementations of the Ethereum node [**109**]. N-version programming is a complementary line of work towards improving fault independence [**18**, **40**], which may be incentivized through methods presented in this work.

For closed BFT systems, methods to manage node diversity have been proposed. Common weaknesses can be avoided at initialization [**81**], and vulnerabilities monitored to evaluate risk as the protocol runs [**64**].

As opposed to simply assuming fault independence in security proofs, the f-out-of-n model of BFT has been expanded to include dependent faults [**34**], and can be generalized through Byzantine adversary structures [**84**].

The remote attestation of configurations has been systematized [**58**] and multiple methods have been proposed. Examples include leveraging trusted platform modules [**42**], verification by third parties [**58**], messaging protocols using challenges [**115**], and secure co-processors [**117**]. Understanding the suitability of each available technique for different configuration aspects of blockchain nodes is considered orthogonal to this work.

2.4 Cryptographic tools

2.4.1 Secret leader election for blockchain

The previously mentioned blockchains elect a secret leader to propose the next block, avoiding targeted attacks. This is not the case in BFT protocols where a public round-robin leader schedule [**130**] [**29**]or a public random schedule as in Ethereum PoS commonly used. In the asynchronous BFT line of work, where crashed leaders and leaders with a slow network can't be differentiated, protocols such as [**10**] [**89**]have every replica act as leaders in parallel and choosing one retrospectively at an $O(n)$ additional communication cost. The different secret leader selection mechanisms that have been proposed are summarized in the following section:

2.4.1.1 Proof of work

Bitcoin [**102**] introduces Proof-of-Work (PoW) for consensus by using a hash function that takes the previous block header, block contents, and iterative nonce as input. By rapidly computing new hashes with new nonces, miners can be randomly selected, weighted by their computiting power. A difficulty parameter decides which hash outputs are small enough to count as a valid PoW for a new block.

2.4.1.2 Verifiable random functions

Micali et al. [**96**] introduces VRFs as a way of generating a random number along with a proof that it was generated honestly. The proof uses a secret and public key pair,

2.4.1.3 Single secret leader election

Single Secret Leader Election (SSLE), introduced by Boneh et al. [**27**] propose a way to elect an anonymous leader until it revels it was chosen, without having to wait for reception of all messages such as with the VRF approach. Catalano et al. [**37**] then improves on the protocol to handle adaptive corruptions and removes the need for elected leaders to re-register after revealing their winning secret.

2.4.1.4 Randomness beacon selection

Randomness beacons are crucial to any consensus protocol randomly selecting nodes to perform a certain role. VRF functions also need a common random input, which is publicized on-chain in Algorand and Ouroboros. Random beacon are also used in Ethereum 2.0 to randomly select the next few leaders in a public way. Without a blockchain that requires more rounds in expectation with protocols like SPURT [**47**] that perform some kind of BFT consensus to agree on the final output.

2.4.1.5 Verifiable delay functions

Verifiable Delay Functions (VDF), introduced by Boneh et al. [**26**], can provide a verifiable random output computed in a predictable amount of time, irrespective of one's computational power or parallelization capabilities. The main advantage they bring for our application is that they don't require any secret values held by participants like a PKI or threshold signature. This can be a way for mobile adversary secure protocols to avoid the overhead of constant secret refreshing.

2.4.2 Public key infrastructure and signature schemes

2.4.2.1 Key management

The common PKI assumption should also hold against a mobile adversary. To achieve this, replicas must be able to regularly renew their keys, even after losing a previous set of keys. This is avoided in the dynamic committee model [**112**] as it instead assumes a steady supply of fresh honest participants each new epoch to keep the f bounded adversary assumption.

2.4.2.2 Broadcast channels

Early work on PSS [**77**] makes use of broadcast channel assumptions. While this can be realistic in networks sharing the same communication medium, it is not possible in point-to-point networks without expensive reliable broadcast primitives.

2.4.2.3 Authenticated point-to-point channels

Authenticated point-to-point channels are commonly assumed in distributed protocols. This is reasonable for practical use in standard adversary models as participants only need to setup the channels once. In the context of a mobile adversary however, this assumption implies additional methods to regain control of the channel after corruptions, or ways to regularly add new channels with all participants in the dynamic committee case. In classical distributed research, this assumption is well suited as the application of local area networks has physical cable links between processors. Mobile faults can therefore not permanently compromise the physical communication links after recovery.

In a setting over the internet, authenticated point-to-point channels imply the existence of a PKI to authenticate messages sent over TCP/IP. Authenticated point-to-point channel assumptions therefore inherit the same issues as PKI in the presence of mobile adversaries.

2.4.2.4 Forward and backward security

One possibility to renew secret keys are intrusion resistant signatures, such as proposed by Dodis et al. [**51**], where keys are regularly refreshed with the help of a base, a separate machine, such that stealing the secret key only enable spoofing signatures for that specific time period. As long as the base and main signer are never simultaneously corrupted, the adversary can't forge future or past signatures by temporarily stealing one's secret key.

The implementation of the base still requires extra overhead however, and could introduce new assumptions.

2.4.2.5 Distributed certification authority

Another option to safeguard public key infrastructure against a mobile adversary is using a distributed certification authority such as COCA [**132**] where frequent key updates are possible. Once again, the implementation of this protocol must have safe communication links itself.

2.4.3 Light clients

2.4.3.1 Non interactive proofs of proof of work

Proving the presence of a specific block in a blockchain can be achieved with proofs of logarithmic size to the chain length, instead of the linearly growing full chain or trusting a third party. NiPoPows [**82**] make this possible by assigning block "levels", superblocks, based on the number of extra zeroes in front of their PoW hash. The key intuition is that the number of such levels scales logarithmically, where all blocks are level 0, half are level 1, a quarter are level 2, and so on. The prover provides the proof π along with the last k blocks. The proof π includes the last m superblocks of every level with enough blocks, along with the lower level blocks narrowing down to the transaction considered. The verifier obtains proofs from multiple provers and chooses the proof with the most superblocks, probabilistically guaranteeing security. The size of π scales logarithmically.

2.4.3.2 FlyClient

FlyClient [**31**] uses probabilistic sampling of blocks according to the distribution maximizing the chance of finding the adversary's incorrect blocks. A merkle mountain range construction linking all blocks together whose root is available at every chain tip. By only providing randomly sampled block headers for the verifier, where only a logarithmic growing number of blocks are required for a vanishing forgery probability, FLyClient achieves even smaller proofs than NiPoPow.

2.4.3.3 Mithril: Stake weighted threshold signatures

In a PoS blockchain, the honest majority assumption can also be leveraged to obtain short threshold signatures, presented in Mithril [**38**]. Instead of requiring k of n signatures as in traditional threshold signatures, a subset of all stake is sampled, becoming a t fraction of all stake threshold. By increasing the sample size the reliability of the signature increases, at the cost of more communication and signature aggregation computation due to having more active participants.

CHAPTER 3

Mobile Crash Byzantine Adaptive Model.

3.1 Introduction

Byzantine Fault Tolerant (BFT) protocols provide a performant and energy-efficient alternative to Proof-of-Work (PoW) consensus for distributed ledgers. Many BFT protocols follow the leader-based design [**29**, **36**, **49**, **130**], which uses a leader to propose values, replaced in case of misbehavior or simply for fairness.

For leader-based protocols, and generally deterministic consensus, security is guaranteed against a static or adaptive Byzantine adversary, who can corrupt up to a threshold of f nodes, and under a synchronous or partially synchronous network to circumvent the Fischer, Lynch, and Paterson (FLP) impossibility result [**56**]. In these network models, an honest leader's message is guaranteed to be delivered within a known upper bound of network delay (after global stabilization time in a partially synchronous network). The bounded network delay enables leader rotations after a timeout.

Mobile DoS attacks. For internet implementations, an adversary may attempt Denial-of-Service (DoS) attacks on leaders to stall the protocol. If leaders are known a priori, then the adversary can target every future leader in each rotation. To highlight the capability of moving the target between leaders, we call such an attack a *mobile DoS* attack. To date, various DoS attacks on deployed blockchains have already been attempted [**76**, **90**], illustrating the need to consider protocols secure in a model fully capturing DoS threats.

While such an attack is a common attack vector, classic security models do not cover it — this attack is orthogonal to the Byzantine threshold or network assumption. As a result, BFT

TABLE 3.1. Comparison of existing works and their system models.

Existing works	System model	*Deterministic consensus*	*No identity re-establishment**	*Partition tolerance possible***	*Models mobile DoS*
[**35**, **74**, **89**, **97**]	Asynchronous network Adaptive/static Byzantine	✗	✓	✓	✓
[**29**, **36**, **130**]	Partially synchronous Adaptive/static Byzantine	✓	✓	✓	✗
[**8**, **73**, **114**]	Mixed faults	✓	✓	✓	✗
[**93**]	Alive-but-corrupt faults	✓	✓	✓	✗
[**9**, **48**, **61**]	Synchronous Adaptive Byzantine	✓	✓	✓	✗
[**23**, **28**, **30**, **106**, **110**]	Mobile Byzantine	✓	✗	✓	✓
[**36**, **77**, **112**, **132**]	Slowly Mobile Byzantine	✓	✗	✓	✗
[**72**, **100**, **107**]	Adaptive Sleepy model	?†	✓	✗	✓
[**9**, **75**, **83**]	Mobile sluggish faults	✓	✓	✗	✓
This work	**MCAB**	✓	✓	✓	✓

* Identity re-establishment is a very strong requirement, often requires manual operations, making it impractical.
** This indicates the possibility of making protocols simultaneously tolerate partition and secure against a given adversary model.
† This is still an open question as per [**107**, Section 6].

protocols under these models either do not tackle such an attack, or only provide an informal discussion on it.

For example, two deployed Proof-of-Stake (PoS) blockchains using provably secure protocols, Cardano and Algorand, seem to consider handling DoS as a design goal: Algorand [**67**] states that *"Algorand is not susceptible to either targeted compromises or DoS attacks"* and Cardano documentation states that Ouroboros Praos prevents DoS attacks [**79**] without proofs. In fact, Ouroboros Praos and Algorand are shown to be secure under an adaptive adversary in a semi-synchronous or weak synchronous network respectively, either of which does not allow an attacker to perform mobile DoS attacks. This highlights the gap between the considered adversary in practice and in theoretical security analysis. A deep and systematic understanding is lacking, including the desired properties required to resist against such attacks.

Existing models and adversaries. Alternative system models (summarised in Table 3.1 and analyzed in §2.2) have been proposed in the literature. However, they either only partially capture the mobile DoS attack, or make it impossible to design protocols with desired features. For example, it is impossible to design a system that is secure in the Sleepy or Mobile Byzantine model in a network that is not synchronous [**107**, **120**]. For the asynchronous network model, deterministic consensus is impossible due to the FLP result [**56**]. Therefore, an adversary model is needed to capture mobile DoS attacks but still allow consensus under different network models, and allow both probabilistic or deterministic guarantees, without contradicting the FLP result [**56**].

Capturing mobile DoS. To model the mobile DoS attack, the adversary always needs the ability to target a new node. This is not the case when the adversary eventually can't target certain honest nodes. For example, the classical adaptive adversary can't corrupt new nodes after f corruptions. Network models that eventually deliver all messages similarly can't model mobile DoS.

The mobile Byzantine adversary in some works [**36**, **77**, **112**, **133**], which we call slow, compromises new sets of nodes periodically, such that the adversary can't corrupt new nodes after f corruptions within one period.

Deterministic consensus. As shown by the FLP impossibility result [**56**], deterministic consensus is impossible in a fully asynchronous network. In the sleepy model [**107**], deterministic protocols remain an open question. As a result, using asynchronous BFT's or the sleepy model's line of work may be unsuitable for applications requiring deterministic guarantees.

Requiring no identity re-establishment. Protocols requiring identity re-establishment need to re-gain control of the identity after the mobile attacker leaves [**30**, **63**, **77**, **106**, **112**]. This includes two common implementations, including physical links and manual operations. For the former, it assumes that the servers are authenticated through physical channels, such that when the attacker leaves at time t', it cannot impersonate as the victim at time $t > t'$. For the latter, it assumes that the victim has the capacity to re-establish its identity at the speed of the

adversary in corrupting a node. This typically requires manual process such as re-registration over the PKI or manually accessing a secure off-line storage.

Partition tolerance. Typically protocols under synchronous networks do not consider partition tolerance; however, partition tolerant protocols can still be secure against some adversaries in synchronous networks. Nonetheless, it is impossible for protocols to be secure under the sleepy model [**107**] and the mobile sluggish fault model [**75**], while tolerating partition.

Our contribution. This work addresses the research gap surrounding the mobile DoS attack with three major contributions. First, we define an adversary model, the *mobile crash adaptive Byzantine (MCAB)* adversary. While the mobile crash adversary solely models mobile DoS attacks, it can be combined with the widely accepted adaptive Byzantine adversary for the targeted applications, i.e., reaching consensus over the Internet. We adapt and extend the concept of roles [**66**] to model consensus, in order to formally analyze the security gap between the MCAB model and the adaptive Byzantine model. Second, we explore and identify the entire design space for maintaining liveness of consensus protocols under the MCAB model through two key properties that we call *abundance* and *concealment*, and analyze their trade-offs. Our general positive results show that either one of those properties are sufficient for liveness in the MCAB model, and can serve as a modular tool for simplifying the analysis of existing and future protocols in the MCAB model. Third, to illustrate the modular utility of our general results, we leverage our findings on case studies: we show that Ourorobors Praos and Algorand are indeed still secure against the MCAB adversary by showing that they satisfy our concealment key property above. We also show how to augment Hotstuff to remain secure against the MCAB adversary via showing its abundance property.

*Formalizing the model and roles.*We formally model consensus by adopting the concept of roles. In addition, to decouple the adversary's capacity from network assumption, we define the capacity of an adversary to corrupt nodes as oracles, allowing our model to be applied to different networks, not only covering different models in the literature but also providing new unexplored scenarios (as shown in Fig 3.1). The MCAB adversary has access to f adaptive Byzantine faults, and c mobile crash faults. Assumed bounds on f and c enable a fine-grained choice of adversarial power for different applications.

Properties for liveness in the MCAB adversary model. We define two properties and prove that either is necessary, and both are sufficient for liveness:

- *Concealment*, inspired by the random secret leaders in blockchains [**21**, **67**] and BFT achieving constant expected rounds [**7**], covers roles performed by nodes whose identities are only revealed after they broadcast the role's messages.
- *Abundance*, inspired by paralleled BFT where every node participates in each step [**89**, **97**, **123**], captures roles performed by more nodes than the adversary's fault capacity f.

We also prove that the safety guarantee of consensus protocols in their original model can be transferred into the MCAB model (Theorem 1 and Theorem 2).

Understanding design trade-offs. To further our understanding on concealment and abundance, we derive lower bounds for the minimum number of communication rounds and message complexity introduced by concealment or abundance (Theorem 5-7). We are then able to present a full picture of the trade-offs between concealed and abundant protocol designs using our lower bounds and applicable existing work.

Case studies with known protocols. We show that Algorand is safe and live in the MCAB adversary model, confirming the informal discussion that Algorand solves consensus against a stronger adversary than the adaptive Byzantine adversary. Hotstuff, on the other hand, loses liveness as it was strictly designed for the adaptive Byzantine adversary.

The rest of this paper is as follows. Section 3.2 introduces the system model. Section 3.3 formalizes the MCAB adversary model and proves that protocol's persistence is maintained against the MCAB adversary. Section 3.4 analyzes necessary and sufficient properties for liveness. Before concluding our work, Section 3.6 leverages our findings on two case studies, Algorand and Hotstuff.

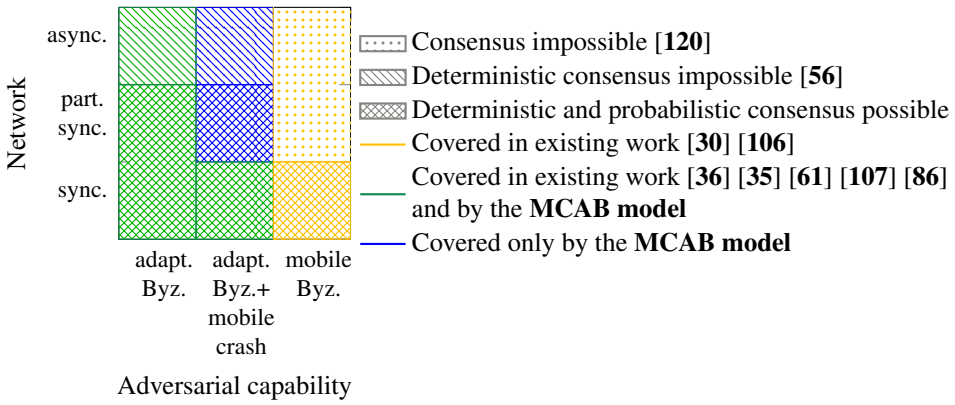


FIGURE 3.1. Landscape of existing models.

3.2 System model and preliminaries

This section introduces the system model, network assumptions, and preliminaries on consensus.

3.2.1 System model

The system consists of a set of n distributed participants, which we call *nodes*, running a pre-defined consensus protocol. In applications over the internet, they are usually online servers. We include scenarios where nodes represent fractions of stake or total hash power by considering the smallest units of stake or hash power as the equivalent to single nodes. For example, a participant controlling $1/3$ of stake can be modeled as $1/3$ of all individual nodes in the system.

Honest nodes follow the specified protocol and send or receive messages according to the network model. However, some nodes may be faulty, which is modelled as an *adversary* with access to different corruption oracles.

Section 3.3 defines the full range of adversarial capabilities we consider.

3.2.2 Network assumptions

Nodes are connected to each other in a network of point-to-point communication links. The three types of assumptions on message delivery are as follows:

Synchrony In synchronous networks, all messages sent between honest nodes are delivered within a known time bound Δ. The order in which they arrive can be determined by the adversary.

Asynchrony In an asynchronous network, message delivery is determined by the adversary, as long as all messages sent between honest nodes are eventually delivered. Messages between honest nodes cannot be dropped or modified.

Partial synchrony Partial synchrony behaves asynchronously until after an unknown time, Global Stabilization Time (GST), after which the network behaves synchronously with a known time bound Δ between honest nodes.

3.2.3 Consensus protocols

We first define the concept of communication steps and negligible functions and present the properties of consensus.

Communication steps To track time in the system, we use a *communication step* counter denoted with $s \in \mathbb{N}_0$ and initialized at 1. As considered in asynchronous networks to count rounds [**97**], all messages are assigned a virtual round number by the system, with the condition that messages with virtual round $s+1$ can only be sent once all messages with virtual round $s-1$ between honest nodes have been delivered. Note that virtual rounds are only used for analysis and not contained in concrete messages. The current communication step s is the highest assigned virtual round. In (partially) synchronous networks, the communication step s is also incremented once the network's delay bound Δ is reached.

Negligible function A negligible function refers to a function $negl(x)$ where for every $y \in \mathbb{N}$, there exists an $z \in \mathbb{N}$ such that $negl(x) < 1/x^y$ for all $x \geq z$.

Message complexity Message complexity is the expected total number of messages generated by honest nodes.

Consensus Blockchain consensus mainly considers two properties, namely *persistence* and *liveness* [**61**]. Intuitively, persistence guarantees that a valid entry in the ledger is consistent across nodes and time, while liveness ensures that new inputs can be added to the ledger.

DEFINITION 1 (Persistence [**61**]). *A consensus protocol satisfies persistence if the following holds. At each communication step s, if an honest node reports a ledger that contains an input v in a block more than $k \geq 0$ blocks away from the end of the ledger, then v will be eventually always be reported in the same position in their ledger by all honest nodes for all communication steps $s' > s$ except with probability $negl(k)$.*

The block depth k starts at $k = 0$ for the most recent one. Persistence is achieved with probability 1 once a block reaches a pre-defined k. In contrast, the probability of persistence in Nakamoto consensus grows asymptotically close to 1 with k.

DEFINITION 2 (t-Liveness [**61**]). *A protocol $\mathcal{P}$ achieves t-liveness if the following holds. At each communication step s, if a valid input v, given to all honest nodes during each communication steps s to $s + t(k)$, then v is reported as k deep by all honest nodes after communication step $s' > s + t(k)$, except with probability $negl(k)$. The number of communication steps t is a known polynomial function of k.*

3.3 Mobile crash adaptive Byzantine adversary model

This section defines adversarial capabilities, presents a framework for analyzing protocols with roles, and shows how the persistence of protocols against the adaptive Byzantine adversary can be transferred to the MCAB model.

3.3.1 Corruption oracles and fault mobility

A *corruption oracle* is an abstract black box representing the adversary's capabilities. It refers to either the Byzantine oracle in Definition 3 or the crash oracle in Definition 4.

DEFINITION 3 (Byzantine oracle $\mathcal{O}_B$). *The Byzantine oracle $\mathcal{O}_{B,f}$ takes queries with an input id and is parameterized by the corruption budget $f \in \{0, ..., n\}$. For simplicity, we denote $\mathcal{O}_{B,f}$ with $\mathcal{O}_B$. For a query at communication step s, if $|\mathcal{B}_s| < f$ for the set $\mathcal{B}_s$ of Byzantine nodes at step s, $\mathcal{O}_B$ adds id to $\mathcal{B}_s$, and outputs the internal state of node id to the adversary. Nodes in $\mathcal{B}_s$ can deviate arbitrarily from the protocol during s. If $|\mathcal{B}_s| \geq f$, $\mathcal{O}_B$ outputs $\perp$.*

DEFINITION 4 (Crash oracle $\mathcal{O}_C$). *The crash oracle $\mathcal{O}_{C,c}$ takes queries with an inputs id and is parameterized by by the corruption budget $c \in \{0, ..., n\}$. For simplicity, we denote $\mathcal{O}_{C,c}$ with $\mathcal{O}_C$. For a query at communication step s, if $|\mathcal{C}_s| < c$ for the set $\mathcal{C}_s$ of crashed nodes, $\mathcal{O}_C$ adds id to $\mathcal{C}_s$. Nodes in $\mathcal{C}_s$ keep their memory and follow the protocol, but outgoing and incoming messages may not deliver during communication step s. If $|\mathcal{C}_s| \geq c$, $\mathcal{O}_C$ outputs $\perp$.*

Blockchains traditionally use an adaptive adversary [**48**, **61**, **67**] that gradually corrupts nodes over time, reflected in our definition by allowing corruption queries at any communication step. Once corrupted, nodes remain under adversarial control. We define adaptive adversaries and obtain the adaptive Byzantine adversary $\mathcal{A}_{AB}$.

DEFINITION 5 (Adaptive adversary). *An adaptive adversary with a corruption oracle $\mathcal{O} \in \{\mathcal{O}_C, \mathcal{O}_B\}$ can query $\mathcal{O}$ any number of times at the start of any communication step s, before nodes send or receive messages during s. A node corrupted by an adaptive adversary at s is corrupted for all $s' > s$.*

Introduced in the context of traditional BFT [**30**, **106**] and proactive secret sharing [**77**, **112**], mobile adversaries also gradually corrupt nodes but can additionally make corrupted nodes honest again. As the protocol progresses, this lets the adversary corrupt more nodes in total while the number of faults at any time remains bounded. We formally define a mobile adversary in Definition 6.

DEFINITION 6 (Mobile adversary). *A mobile adversary with a corruption oracle $\mathcal{O} \in \{\mathcal{O}_C, \mathcal{O}_B\}$ can query $\mathcal{O}$ any number of times at the start of any communication step s, before any nodes send or receive messages. At the start of s, before querying $\mathcal{O}$, a mobile adversary is also able to* cure *a node it has corrupted so that it becomes honest again, and receives all messages previously sent to it.*

By limiting the oracle queries and curing of nodes at a communication step s before any nodes send or receive messages, the possible number of faulty nodes at any communication step s is not higher than the corruption budget of the adversary, as in Buhrman et al. [**30**]. For example, by replacing a faulty node during s with another one, both count as a fault during s.

Using the previous definitions, we can define the mobile crash fault adaptive Byzantine adversary $\mathcal{A}_{MCAB}$.

DEFINITION 7 (Mobile crash adaptive Byzantine adversary). *The mobile crash adaptive Byzantine adversary $\mathcal{A}_{MCAB}$ is mobile with the crash oracle $\mathcal{O}_c$ and corruption budget c, and is adaptive with the Byzantine corruption oracle $\mathcal{O}_B$ and corruption budget f.*

Nodes in the Byzantine set cannot be replaced or removed without some recovery mechanism for secret keys and authenticated communication channels. In contrast, crashed nodes go back online if they are not targeted again in communication step $s + 1$ and receive all previous pending messages as in the sleepy model [**107**]. As a result $\mathcal{A}_{MCAB}$ has adaptive Byzantine and mobile crash faults.

3.3.2 Modelling roles in consensus

To analyze protocols in our proposed model, we follow and extend the idea of a role/node framework for general multi-party computation introduced in YOSO [**66**] for consensus. A protocol consists of atomic tasks, performed by different roles at every communication step. Roles, formalized in Definition 8, are then assigned to nodes with a role assignment mechanism as the protocol runs. Known protocols described in this framework are provided in Section 3.6.

DEFINITION 8 (Role). *Roles are a subset of the protocol's sequence of tasks, whose input values come from received messages and local memory and whose outputs are one or more messages communicated to one or more other participants and updating local memory.*

- *Role assignment: A role $\mathcal{R}$ is assigned at communication step s to a set of nodes, the committee $\mathcal{M}_s$, by a role assignment oracle $assignRoles_{\mathcal{R}}(s, \lambda)$ with security parameter λ.*
- *Role existence: A role $\mathcal{R}$ exists at a communication step s if the protocol requires at least one node to perform the role's tasks at s. If $\mathcal{R}$ does not exist at step s, then $\mathcal{M}_s = \emptyset$.*

3.3.3 Persistence in the MCAB model

We prove that protocols with persistence against $\mathcal{A}_{AB}$ can also achieve persistence against $\mathcal{A}_{MCAB}$.

THEOREM 1 (Persistence against MCAB). *If a protocol P achieves persistence against $\mathcal{A}_{AB}$ with f Byzantine corruptions, then P achieves persistence against $\mathcal{A}_{MCAB}$ with c mobile crash and $f - c$ Byzantine corruptions.*

PROOF. We prove by contradiction. Consider a protocol P that achieves persistence against $\mathcal{A}_{AB}$, but not against $\mathcal{A}_{MCAB}$. This means that, with probability larger than $negl(k)$, a stable input $v' \neq v$ is reported at the same position as v in the ledger by an honest node at communication step s', where $s' > s$. The adversary may let nodes selected as crashes to send messages supporting this persistence violation, such that in the worst case, $\mathcal{A}_{MCAB}$ has $f - c + c$ nodes with Byzantine behaviour. $\mathcal{A}_{MCAB}$ must be able to create this persistence violation with probability greater than $negl(k)$ using only these $f - c + c$ nodes. This forms a contradiction as P achieves persistence with up to f Byzantine nodes, and $\mathcal{A}_{MCAB}$ must be able to create a persistence violation controlling only up to f nodes with Byzantine behaviour. □

3.4 Liveness in the MCAB model

In this section, we explore security requirements for a protocol to maintain liveness against the MCAB adversary. We first show that protocols secure in an asynchronous network are always secure in the MCAB model. We then introduce and prove the necessary and sufficient conditions for liveness against the MCAB adversary.

Definitions in this section consider the following notations: A consensus protocol P with persistence and t-liveness against an adversary $\mathcal{A}_{AB}$ with access to the Byzantine oracle $\mathcal{O}_B$ and corruption budget f; An adversary $\mathcal{A}_{MCAB}$ with access to the crash oracle $\mathcal{O}_C$ with corruption budget c and to the Byzantine oracle $\mathcal{O}_B$ with corruption budget $f - c$; The set $\mathcal{N}_s$ of n_s nodes running P at a communication step s, identified by id where $id \in [1, ..., n_s]$; The Byzantine set $\mathcal{B}_s$, the crashed set $\mathcal{C}_s$, and the honest set $\mathcal{H}_s$ where $\mathcal{H}_s = \mathcal{N}_s \setminus (\mathcal{B}_s \cup \mathcal{C}_s)$ at a communication step s; A role $\mathcal{R}$ in P, assigned to nodes in the set $\mathcal{M}_s$ at a communication step s; The set $\mathcal{S}_\mathcal{R} = \{\mathcal{R}_j | j \in [1, p]\}$ of all roles in P. Each $\mathcal{R}_j$ where $j \in [1, p]$ is assigned to nodes in the set $\mathcal{M}_{j,s}$ at a communication step s; The role assignment oracle for $\mathcal{R}$ denoted as $assignRoles_\mathcal{R}$.

3.4.1 Asynchronous networks

We prove that asynchronous BFT protocols are also secure in our model. In partial synchrony, $\mathcal{A}_{MCAB}$ is hence only different from $\mathcal{A}_{AB}$ when considering liveness in the synchronous period after GST.

THEOREM 2 (Asynchronous BFT is secure against the MCAB adversary). *If a protocol P achieves persistence and t-liveness (after t asynchronous rounds) in an asynchronous network against $\mathcal{A}_{AB}$ with f adaptive Byzantine corruptions, then it achieves persistence and t-liveness against $\mathcal{A}_{MCAB}$ in a synchronous network with c mobile crash corruptions and $f - c$ adaptive Byzantine corruptions.*

PROOF. To show that persistence is maintained against $\mathcal{A}_{MCAB}$ by any protocol with persistence against $\mathcal{A}_{AB}$, see Theorem 1. We prove t-liveness is maintained by contradiction.

Consider a protocol P with t-liveness in an asynchronous network against $\mathcal{A}_{AB}$, but not against $\mathcal{A}_{MCAB}$ in a synchronous network. There therefore exists a valid input v given to all honest nodes in $\mathcal{H}_s$ at communication step s, not considered stable for any nodes in $\mathcal{H}_{s'}$ for all communication steps $s' > s + t$. This means that the $n - f$ nodes in each set $\mathcal{H}_{s+i}$ where $i \in [0, t)$, who receive all previous honest messages and deliver new messages within a known delay Δ, don't achieve stability for v for at communication step $s + t$. However, P has t-liveness in an asynchronous network against $\mathcal{A}_{AB}$, and any input becomes stable after t asynchronous rounds between $n - f$ honest nodes. As t asynchronous rounds can always model t synchronous communication steps by definition, this is a contradiction. □

3.4.2 Properties for liveness

The rest of our analysis focuses on liveness in a synchronous network or the synchronous period of a partially synchronous setting when transferring protocols live against $\mathcal{A}_{AB}$ to $\mathcal{A}_{MCAB}$.

Inspired by protocols with secret block proposers [**48**, **61**, **67**] and protocols where every node participates at every step [**89**, **97**, **123**], we derive two properties using the role framework of Section 3.3.2, abundance and concealment.

A (r, q, t)-essential role is first defined to exclude superfluous roles for the sake of counter-examples It is parameterized by the minimum number r of assignments to the role that need to be honestly executed at least q times out of t communication steps for the t-liveness of the protocol.

Informally, essential roles represent the steps in a protocol crucial to its proper execution, while non-essential roles could be skipped without consequence.

DEFINITION 9 ((r, q, t)-essential). *Every role $\mathcal{R}_j$ with $j \in [1, p]$ in the set $\mathcal{S}_\mathcal{R}$ is* (r_j, q_j, t)-essential *where $r_j, q_j \in \mathbb{N}$ if the following holds. At communication step s, for a valid input w given to all nodes in every $\mathcal{H}_{s+i}$ where $i \in [0, t)$ in P, if there exists $l \in [1, p]$ such that $|\mathcal{H}_{s+i} \cap \mathcal{M}_{l,s+i}| < r_l$ more than $t - q_l$ times in t communication steps $s + i$ where*

$i \in [0, t)$, then w is stable in the ledger after t communication steps with probability lower than $1 - negl(t)$.

Conversely, if for every $j \in [1, p]$, $|\mathcal{H}_{s+i} \cap \mathcal{M}_{j,s+i}| \geq r_j$ more than q_j times in t communication steps $s + i$ where $i \in [0, t)$, then w is stable in the ledger after t communication steps except with probability $negl(t)$.

We first define abundance, where the role is assigned to a sufficient number of nodes to tolerate all corruptions.

DEFINITION 10 (Abundance). *The (r, q, t)-essential role $\mathcal{R}$ is* abundant *if, for at least q out of any t communication steps $s + i$ where $i \in [0, t)\}$, we have $|\mathcal{M}_{s+i}| \geq\ f + r$.*

The second property is concealment, where the identity of the assigned nodes is secret and $\mathcal{A}_{MCAB}$ can only randomly guess which nodes to target. This ensures that the adversary can't reliably stall the protocol.

Concealment is defined by considering the adversary's win probability in a game where the adversary tries to select the nodes assigned to a role.

DEFINITION 11 (Concealment). *The non-abundant (r, q, t)-essential role $\mathcal{R}$ is concealed if it always wins the following experiment except with probability negligible in t. The* concealment *experiment between the (r, q, t)-essential role $\mathcal{R}$ and the adversary $\mathcal{A}_{MCAB}$ is defined as t consecutive runs of the following game in all communication steps $s + i$ for some s with $i \in [0, t)$. At the start of the game, up to $f - c$ nodes are in the Byzantine set $\mathcal{B}_{s+i}$ for $i \in [0, t)$.*

- *Learning phase: $\mathcal{A}_{MCAB}$ chooses any number of learning rounds. For each learning round, $\mathcal{A}_{MCAB}$ picks $l \notin [s, s + t)$, sends it to $\mathcal{R}$ who calls $assignRoles_{\mathcal{R}}(l, \lambda)$ to obtain $\mathcal{M}_l$, and sends $\mathcal{M}_l$ to $\mathcal{A}_{MCAB}$.*
- *Challenge phase: $\mathcal{A}_{MCAB}$ initiates the challenge phase, calling $\mathcal{O}_C$ any number of times and obtaining the set of crashed nodes $\mathcal{C}_{s+i}$. $\mathcal{R}$ obtains $\mathcal{M}_{s+i}$ with $assignRoles_{\mathcal{R}}(s + i, \lambda)$, and $\mathcal{A}_{MCAB}$ wins the game if $|\mathcal{M}_{s+i} \setminus (\mathcal{B}_{s+i} \cup \mathcal{C}_{s+i})| < r$,*

which we denote by $\mathbf{d}_i$. We also denote by $\mathbf{d}_i$ the indicator random variable for the event $\mathbf{d}_i$, which equals 1 if $\mathbf{d}_i$ occurs and 0 otherwise.

If event $\mathbf{d}_i$ occurs for more than $t-q$ values of $i \in [0,t)$, $\mathcal{A}_{MCAB}$ wins the experiment, denoted by the event $\mathbf{D}$. Note that $\mathcal{A}_{MCAB}$ can add Byzantine nodes with $\mathcal{O}_B$ each communication step $s+i$ if $|\mathcal{B}_{s+i}| < f - c$.

The (r,q,t)-essential concealed role $\mathcal{R}$ is ideally-concealed *if, for all i where $\mathcal{R}$ exists, the indicator random variables $\mathbf{d}_i$ are i.i.d and $\Pr[\mathbf{d}_i] - \sum_{j=0}^{r} \binom{\mathcal{M}}{j} (\frac{f}{n})^j (1-\frac{f}{n})^{r-j} < negl(\lambda)$.*

We define protected roles as roles that can't be reliably targeted by the additional capabilities of $\mathcal{A}_{MCAB}$ by being abundant or concealed.

DEFINITION 12 (Protected). *The (r,q,t)-essential role $\mathcal{R}$ is protected if it is abundant or concealed.*

3.4.3 Necessity and sufficiency for liveness

We show that each role of a protocol being protected is sufficient and necessary to remain secure against $\mathcal{A}_{MCAB}$.

With Lemma 1 we show the necessity. Intuitively this is due to the adversary being able to target all roles that are not protected with mobile crashes.

LEMMA 1 (Necessity for liveness). *For any consensus protocol P, t-liveness against $\mathcal{A}_{MCAB}$ is impossible if there exists one (r,q,t)-essential role that is not protected.*

PROOF. We prove by contradiction. Say an (r_j, q_j, t)-essential role $\mathcal{R}_j$ for P is not protected but P achieves t-liveness against $\mathcal{A}_{MCAB}$.

As $\mathcal{R}$ is not protected, for t communication steps $s+i$ where $i \in [0,t)$, the event $(|\mathcal{H}_{s+i} \cap \mathcal{M}_{j,s+i}| < r_j)$ can happen more than $t - q_j$ times with probability larger than $negl(t)$. Due to the (r_j, q_j, t)-essential property of $\mathcal{R}_j$, this means that a valid input w given to every $\mathcal{H}_{s+i}$

where $i \in [0, t)$ is not stable in the ledger after t communication steps with probability larger than $negl(t)$. This contradicts the assumed t-liveness. □

Lemma 2 shows the sufficiency of protected roles for liveness.

LEMMA 2 (Sufficiency for liveness). *A consensus protocol P with persistence and t-liveness against $\mathcal{A}_{AB}$ always achieves t-liveness in $\mathcal{A}_{MCAB}$ if all node actions are implemented through protected (r, q, t)-essential roles and their respective role assignment mechanism.*

PROOF. We prove by contradiction. Let P be a consensus protocol with persistence and t-liveness for $\mathcal{A}_{AB}$ but not for $\mathcal{A}_{MCAB}$. Let $\mathcal{S}_{\mathcal{R}} = \{\mathcal{R}_j | j \in [1, p]\}$ be the set of all p (r_j, q_j, t)-essential roles in P, which are protected, and let $\mathcal{H}_s$ be the set of honest nodes at communication step s. For every role $\{\mathcal{R}_j \in \mathcal{S}_{\mathcal{R}} | j \in [1, p]\}$, consider the following.

Since $\mathcal{R}_j$ is protected, in t communication steps $s + i$ where $i \in [0, t)$, $|\mathcal{H}_{s+i} \cap \mathcal{M}_{j,s+i}| \geq r_j$ occurs at least q_j times with probability $1 - negl(t)$. By the r_j, q_j, t-essential property of every $\mathcal{R}_j, j \in [1, p]$, any valid input v given to every $\mathcal{H}_{s+i}$ where $i \in [0, t)$ is stable in the ledger after t communication steps except with probability negligible in t. By definition of t-liveness, P also has t-liveness in $\mathcal{A}_{MCAB}$, a contradiction.

□

Finally we show that, for protocols to maintain their security properties against $\mathcal{A}_{MCAB}$, their roles must be protected.

THEOREM 3 (Necessary and sufficient for liveness). *Consider a consensus protocol P that achieves t-liveness against $\mathcal{A}_{AB}$. For P to have t-liveness against $\mathcal{A}_{MCAB}$, it is necessary and sufficient that all node actions can be defined through protected roles and their respective role assignment mechanism.*

PROOF. This statement follows from Lemma 2 and Lemma 1. □

3.5 Performance trade-offs in the MCAB model

In this section we derive bounds and leverage existing works to analyze the trade-offs between abundance and concealment, summarized in Table 3.1.

3.5.1 Lower bounds

The lower bound on t-liveness due to an abundant role follows from Definition 9.

THEOREM 4 (Abundance lower bound on t-liveness). *For a consensus protocol P_a with t_a-liveness against $\mathcal{A}_{MCAB}$ containing an abundant (r_a, q, t_a)-essential role $\mathcal{R}_a$, we always have $t_a \geq q$.*

PROOF. Due to the (r_a, q, t_a)-essential property as defined in Definition 9, t_a-liveness requires at least $t_a \geq q$. □

We show that using an abundant essential role requires at least $\Omega(n)$ message complexity, due to Definition 10. When every role is abundant, each node performing a role sends its message to the nodes assigned to the next abundant role, obtaining $\Omega(n^2)$ message complexity.

THEOREM 5 (Abundance lower bound on message complexity). *The minimum message complexity of a consensus protocol P_a achieving t_a-liveness against $\mathcal{A}_{MCAB}$ containing an abundant (r_a, q, t_a)-essential role $\mathcal{R}_a$, which sends at least one message, is $\Omega(n)$. If every role in P_a is abundant, the message complexity of P_a is $\Omega(n^2)$.*

PROOF. We prove by contradiction. Say an (r_a, q, t_a)-essential role $\mathcal{R}_a$, which sends one message in a consensus protocol P_a, is abundant but the message complexity of P_a is less than $\Omega(n)$. For $\mathcal{R}_a$, we have $|\mathcal{M}| \geq\ f + r_a$ at least q times in t_a communication steps since it is abundant. Each node in $\mathcal{M}$ sends at least one message by definition. At least $f + r_a$ messages are therefore created in communication steps where $\mathcal{R}_a$ exists. As f scales with n, this implies $\Omega(n)$ message complexity, a contradiction.

We prove by contradiction if every role in P_a is abundant. Say an (r_a, q, t_a)-essential role $\mathcal{R}_a$, which sends messages to the next abundant $(r_{a'}, q', t_{a'})$-essential role $\mathcal{R}_{a'}$ in a consensus protocol P_a, is abundant but the message complexity of P_a is less than $\Omega(n^2)$. For $\mathcal{R}_a$, we have $|\mathcal{M}| \geq f + r_a|$ at least q times in t_a communication steps since it is abundant. Each node in $\mathcal{M}$ sends at least $\Omega(n)$ messages to the $|\mathcal{M}'| \geq f + r_{a'}|$ nodes assigned to $\mathcal{R}_{a'}$. At least $(f + r_a)(f + r_{a'})$ messages are sent in communication steps where $\mathcal{R}_a$ exists and is followed by $\mathcal{R}_{a'}$. As f scales with n, this implies $\Omega(n^2)$ message complexity, a contradiction. □

To lower bound t-liveness when introducing a concealed role, we use the Chernoff bound to obtain t given its parameters and the failure probability. Note that the bound on t is optimistic due to our looser approximations of the Binomial distribution tail bound, and a tighter bound could give a slightly higher t. If $\epsilon \to 0$, $t \to \infty$, and if $p = 0$, $t \geq q$ matching the abundant case.

THEOREM 6 (Concealment lower bound on t-liveness). *For a consensus protocol P_c achieving t-liveness with failure probability ϵ containing a concealed (r, q, t)-essential role $\mathcal{R}_c$, we always have $t \geq q + \nu(q, \epsilon)$ where $\nu(q, \epsilon) > 0$ and ϵ is the liveness error probability.*

PROOF. For a protocol using $\mathcal{R}_c$, t-liveness has a failure probability ϵ. To get the lowest t, $\mathcal{R}_c$ is ideally-concealed and for all $i \in [0, t)$, we denote the i.i.d. $\Pr[\mathbf{d}_i]$ as p from the game in Definition 12. We can therefore consider a Binomial distribution with t trials, p as the success probability, and X as the number of successes. We use the tail bound on Binomial distributions [**16**, Lemma 4.7.2] to obtain a lower bound on t, parameterized by ϵ, p, and q. Using the tail bound to bound the maximal number of successes $t - q$, we get

$$\Pr[X > t - q] = \epsilon \geq \frac{1}{\sqrt{8t\lambda(1-\lambda)}} 2^{-tB(\lambda,p)} \geq \frac{1}{\sqrt{2t}} 2^{-tB(\lambda,p)}.$$

where $\lambda = \frac{t-q}{t}$ and $B(\lambda, p) = \lambda \log \frac{\lambda}{p} + (1-\lambda) \log \frac{1-\lambda}{1-p}$. Using the inequality $\log x \leq x - 1$ where $x \geq 0$, we loosen the bound and derive the following inequality :

$$\log \frac{1}{\epsilon} \leq \frac{1}{2} \log(2t) + tB(\lambda, p)$$
$$0 \leq \frac{t^2}{p} + t(\log \epsilon - \frac{2q}{p} - 1) + \frac{q^2}{1-p} + \frac{q^2}{p}$$

Solving for t assuming $t > 0$ we obtain the lower bound:

$$t \geq p \log \frac{1}{\epsilon} + q + \frac{p}{2}(1 + \sqrt{(\log \epsilon - \frac{2}{p} - 1)^2 - \frac{4q^2}{p}(\frac{1}{1-p} + \frac{1}{p})})$$

Since $q > 0$, and $p, \epsilon \in [0, 1]$, the terms are all positive.

We therefore have $t \geq q + \nu(q, \epsilon)$ where $\nu(q, \epsilon) > 0$ and ϵ is the liveness error probability. □

As a concealed role could be assigned to only one node, the lower bound on message complexity comes from the requirement that all $n - f$ honest nodes receive the value to decide upon, implying at least $\Omega(n)$ message complexity.

THEOREM 7 (Concealment lower bound: message complexity). *The minimum message complexity of a consensus protocol P_c amongst n nodes and f faults achieving t-liveness containing a concealed (r, q, t)-essential role $\mathcal{R}_c$ is $\Omega(n)$.*

PROOF. We have $|\mathcal{M}| < f + r$ all communication steps and $|\mathcal{M} \setminus (\mathcal{B} \cup \mathcal{C})| \geq r$ at least q times in t communication steps. As a result, there are at least r messages sent in the communication steps where $\mathcal{R}_c$ exists. However by definition, all $n - f$ honest nodes must receive the input reported in the same position in the ledger, requiring at least $n - f$ messages for P_c to achieve persistence. These $n - f$ messages must be sent by a role in P_c and its message complexity is $\Omega(n)$. □

3.5.2 Discussion on trade-offs

Communication complexity and rounds It is proven that deterministic consensus requires at least $f+1$ rounds in the worst case with f adaptive crash faults [**12**], and $O(n^2)$ communication complexity [**53**]. These results apply to deterministic fully abundant protocols. In contrast, with randomised consensus, the number of rounds to reach an agreement is probabilistic and is dependent on the error probability as a system parameter (Theorem 6).

A protocol using concealment only requires $O(n)$ message complexity (Theorem 7), enabling subquadratic communication complexity similarly to Algorand [**39**] and Abraham et al. [**6**].

After-the-fact-removal However, as shown by Abraham et al. [**6**], subquadratic complexity is only possible without after-the-fact-removal, which means that the adversary is not able to stop messages already sent by a node it corrupts. With after-the-fact-removal, the adversary is able to “cheat" the game defined in Definition 11 by modifying it such that in the learning phase, it can pick the challenge phase’s communication steps. This implies that no role can be concealed, allowing only abundance. The communication complexity lower bound $O(n^2)$ with after-the-fact-removal [**6**] then matches our derived bound in Theorem 5.

Consider the additional cases where after-the-fact removal is allowed for one type of fault only:

- After-the-fact removal adaptive Byzantine faults only, not mobile crashes: while the adversary hasn’t selected all adaptive Byzantine faults, a protocol with subquadratic communication complexity cannot achieve consensus [**6**]. However,if the adversary has selected its maximum allowed number of adaptive faults, the adversary has no after-the-fact removal ability anymore, and concealment is possible to maintain liveness.
- After-the-fact removal mobile crashes only, not adaptive Byzantine: After-the-fact-removal mobile crashes can perpetually target concealed committees, and only abundance is possible.

Sources of randomness While the implementation of abundant roles has no additional overhead, concealed roles require a mechanism to secretly assign roles to nodes. This line of research has notable examples such as PoW mining [**61**], Verifiable Random Function (VRF) selection [**96**], and Single Secret Leader Election (SSLE) [**27**]. When the the size $|\mathcal{M}|$ of the role's committee is random, rounds where not enough nodes are assigned may be wasted, but achieving deterministic $|\mathcal{M}|$ is expensive.

Probabilistic $|\mathcal{M}|$. A role assignment with probabilistic $|\mathcal{M}|$ assigns a number of nodes sampled from a certain probability distribution, usually such that $\mathbb{E}[|\mathcal{M}|] > r$. Since $|\mathcal{M}|$ is probabilistic, it is still possible to occasionally have $|\mathcal{M}| < r$ without interference from the adversary. This can add additional communication rounds to terminate the protocol, leading to higher values of t for the t-liveness guarantee. For example, VRF random assignments such as in Algorand [**67**] secretly select nodes, satisfying concealment, with a probabilistic expected number of nodes in one committee. It can be the case however, that a committee in Algorand does not reach quorum due to not enough nodes being selected.

Deterministic $|\mathcal{M}|$. A role assignment with deterministic $|\mathcal{M}|$ assigns a fixed number of randomly chosen nodes. Since $|\mathcal{M}| \geq r$ is guaranteed, only by winning the game defined in Definition 12 can $|\mathcal{H} \cap \mathcal{M}| < r$ still happen. SSLE introduces a primitive to achieve such a design.

However, in the only existing protocol with adaptive security to our knowledge [**37**], SSLE runs with additional communication between nodes which has at least $O(n^2)$ message complexity for n synchronous broadcasts. It should also maintain its security properties in the MCAB model, which seems possible at first glance with similar arguments as in Theorem 1 and due to the abundant all-to-all communications.

Random beacon. Concealed roles inherently require a trusted source of randomness for their role assignment. This has been achieved in various ways like an initial trusted nonce for Bitcoin [**61**] or Algorand [**67**] that is updated as the protocol runs. Distributed random beacons that uses either consensus [**47**] or trusted setups [**25**] are also a way to agree on a common

TABLE 3.2. Comparison of abundant and concealed roles. We consider protected (q, r, t)-essential roles with the positive factors $\mu(\epsilon), \nu(q, \epsilon) > 0$ depending on q and the acceptable error probability for liveness ϵ.

	t-liveness bound	Message complexity	No random beacon	Allow after--the-fact--removal	Load balancing
Abundant and deterministic	$t \geq f + 1$	$\Theta(n^2)$	✓	✓	✓
Abundant and randomized	$t \geq q + \mu(\epsilon)$	$\Theta(n^2)$	✗	✓	✓
Concealed with determin. $\lvert\mathcal{M}\rvert$	$t \geq q + \nu(q, \epsilon)$	$O(n^2)$	✗	✗	✗
Concealed with random $\lvert\mathcal{M}\rvert$	$t \geq q + \nu(q, \epsilon)$	$\Theta(n)$	✗	✗	✗

source of randomness. However, they may leak information, increasing $\Pr[\mathbf{d}]$ in Definition 12 and t for t-liveness which must be analyzed properly.

Previous work has also shown the benefit of load balancing [**46**, **60**, **123**], which distributes computation and communication workload to nodes, avoiding performance bottlenecks and obtain higher throughput akin to abundant roles.

Table 3.5.2 summarizes the performance trade-offs between the variants of abundance and concealment.

3.6 Case studies: existing protocols in the MCAB model

This section leverages our theorems to analyze the security of Ouroboros [**20**, **21**], Algorand [**39**, **67**] and Chained Hotstuff [**130**] against $\mathcal{A}_{MCAB}$. We briefly summarize their protocol flow and show Chained Hotsuff's lack of liveness and Algorand's security against $\mathcal{A}_{MCAB}$. Their respective full papers provide formal descriptions and proofs of security against $\mathcal{A}_{AB}$. We additionally present a modified illustrative version of Chained Hotstuff maintaining its liveness against $\mathcal{A}_{MCAB}$ with abundance, and discuss how to leverage existing work on fallback protocols for $\mathcal{A}_{MCAB}$.

3.6.1 Ouroboros Praos

Ouroboros Praos is an adaptively secure PoS protocol where nodes regularly compute a VRF output to check eligibility to be a block proposer. The chain selection rule followed by honest nodes allows them to converge to a consistent view of the blockchain.

Only one role is needed to describe Ouroboros Praos, the slot leader role where a node extends its view of the canonical chain with a new block and broadcasts it to the network. Arguments for Ouroboros' security in the MCAB model are analogous for the adaptively secure Snow White and other similar protocols with a single leader role selected randomly and secretly, for example using a VRF or PoW lottery.

Roles in Ouroboros Praos

- Slot leader role $\mathcal{R}_{sl}$: Create a block of transactions and gossip it to the network. The r value is 1.

Committees of roles

- Committee of role $\mathcal{R}_{sl}$: contains a random number of slot leaders, in the deployed protocol [**79**], we have expected value $\mathbb{E}\left[\tau_{sl}\right] = 0.05$.

Role assignment

The role assignment oracle $assignRoles$ in Ouroboros Praos is implemented via the VRF selection mechanism. Nodes are selected to be slot leaders with a parameterized probability weighted by stake. The inputs to the VRF check are their secret key, their number of coins, the output of the on-chain random beacon, and the τ parameter determining the expected number of selected nodes.

To prove the security of Ouroboros Praos against $\mathcal{A}_{MCAB}$, we simply show that the role in Ouroboros Praos satisfies the concealment requirements of Definition 11, implying security against $\mathcal{A}_{MCAB}$ by Theorem 3 and Theorem 1.

COROLLARY 1 (Security of Ouroboros Praos). *Ouroboros Praos achieves t-liveness and persistence against $\mathcal{A}_{MCAB}$ with $f + c \leq n/2$ where f is the number of adaptive Byzantine faults, c is the number of mobile crash faults, and n is the number of total nodes.*

PROOF. Role $\mathcal{R}_{sl}$ is concealed since the assigned nodes are not public until the blocks are broadcast, and the probability of $\mathcal{A}_{MCAB}$ guessing all proposers is smaller than 1. As per the full analysis of Ouroboros Praos [**20**], the probability that the adversary influences the source of randomness drop exponentially with the number of honest blocks in an epoch, and therefore adds a probability negligible in the security parameter for the adversary to guess assignments. This means $\mathcal{R}_{sl}$ is even ideally-concealed. □

3.6.2 Algorand

Nodes in Algorand [**67**] regularly obtain a pseudorandom number with a Verifiable Random Functions (VRF) whose randomness is verifiable with the node's public key. Computing the VRF output is done locally, allowing nodes to check eligibility to propose blocks and participate in BA⋆ [**67**], a Byzantine Agreement algorithm, without revealing their identity. Once block proposers broadcast a block, nodes are selected for each step of BA⋆ to eventually finalize the block with the lowest VRF output, or an empty block in bad network conditions.

We obtain three roles comprising Algorand: proposing a block, performing a step in its own Byzantine Agreement algorithm BA⋆, and finalizing a block in the last step of BA⋆.

Roles in Algorand

- Block proposer role $\mathcal{R}_{\texttt{PROPOSER}}$: Create a block of transactions and gossip it to the network. The r value is 1.
- Voter in the binary BA⋆ algorithm, step $i \in [1, \texttt{MaxSteps}]$ $\mathcal{R}^i_{\texttt{STEP}}$: The CommitteeVote function checks if one is selected for the next step's role, and is part of the role assignment mechanism of Algorand. The r for $\mathcal{R}_{\texttt{PROPOSER}}$ is $T_{\texttt{STEP}} \cdot \tau_{\texttt{STEP}} = 0.685 \cdot 2000 = 1370$. We refer to Algorithm 8 in [**67**] for the specific operations performed in each step i of BA⋆.

- Voter in the binary BA$\star$ algorithm, final step $\mathcal{R}_{\texttt{FINAL}}$: Perform final step of BA$\star$. The r for $\mathcal{R}_{\texttt{FINAL}}$ is $T_{\texttt{FINAL}} \cdot \tau_{\texttt{FINAL}} = 0.74 \cdot 10000 = 7400$.

Since the parameters for the role assignment mechanism and the voting threshold per BA$\star$ step are the same, we analyze them together indexed with i. The last step denoted `FINAL` has slightly different parameters and is therefore considered separately.

Committees of roles

- Committee of role $\mathcal{R}_{\texttt{PROPOSER}}$: contains a random number of block proposers with expected value $\mathbb{E}\left[\tau_{\texttt{PROPOSER}}\right] = 26$.
- Committee of role $\mathcal{R}^{j}_{\texttt{STEP}}$: contains a random number of BA$\star$ members for BA$\star$'s step j with expected value $\mathbb{E}\left[\tau_{\texttt{STEP}}\right] = 2000$.
- Committee of role $\mathcal{R}_{\texttt{FINAL}}$: contains a random number of final BA members with expected value $\mathbb{E}\left[\tau_{\texttt{FINAL}}\right] = 10000$.

Role assignment

The role assignment oracle $assignRoles$ in Algorand is implemented via the VRF selection mechanism. Nodes are selected for different roles with a parameterized probability weighted by stake. The inputs to the VRF check are their secret key, their number of coins, the current on-chain seed, and the τ parameter determining the expected number of selected nodes.

To prove the security of Algorand against $\mathcal{A}_{MCAB}$, we simply show that every role in Algorand satisfies the concealment requirements of Definition 11.

COROLLARY 2 (Security of Algorand). *Algorand achieves t-liveness and persistence against $\mathcal{A}_{MCAB}$ with $f + c \leq n/3$ for f adaptive Byzantine faults, c mobile crash faults, and n total nodes at any time.*

PROOF. $\mathcal{R}_{\texttt{PROPOSER}}$ is concealed since the assigned nodes are not public until the blocks are broadcast, and the probability of $\mathcal{A}_{MCAB}$ randomly guessing all proposers is smaller than 1. As per the full version of Algorand [**67**], probability of guessing the next selection seeds drop exponentially with the number of blocks committed in a synchronous period, and

therefore adds a probability negligible in the security parameter for the adversary to guess assignments. This means $\mathcal{R}_{\texttt{PROPOSER}}$ is even ideally-concealed.

$\mathcal{R}^j_{\texttt{STEP}}$ is concealed for all j since the assigned nodes aren't public until the votes are broadcast, and the probability of $\mathcal{A}_{MCAB}$ guessing committee members for any s such that $\mathcal{R}_{\texttt{STEP}} \cap \mathcal{H}_s < T_{\texttt{STEP}} \cdot \tau_{\texttt{STEP}} = 1370$ is smaller than 1. As for $\mathcal{R}_{\texttt{PROPOSER}}$, $\forall j, \mathcal{R}^j_{\texttt{STEP}}$ is ideally concealed.

$\mathcal{R}_{\texttt{FINAL}}$ is concealed since the assigned nodes aren't public until the votes are broadcast, and the probability of $\mathcal{A}_{MCAB}$ guessing committee members for any s such that $\mathcal{R}_{\texttt{FINAL}} \cap \mathcal{H}_s < T_{\texttt{FINAL}} \cdot \tau_{\texttt{FINAL}} = 7400$ is smaller than 1. As for $\mathcal{R}_{\texttt{PROPOSER}}$, $\mathcal{R}_{\texttt{FINAL}}$ is ideally concealed.

All roles in Algorand are concealed, and by Lemma 3, Algorand achieves t-liveness and persistence in $\mathcal{A}_{MCAB}$. □

3.6.3 Chained Hotstuff

Chained Hotstuff's [**130**] protocol flow is as follows: a leader collects signatures on the last proposed block and forms a Quorum Certificate (QC) if the threshold is met. It broadcasts the QC along with its own proposed block, which nodes verify before sharing their signature with the next leader. This process repeats and, to guarantee safety and liveness, blocks need three sequential QCs to be finalized. If a threshold of nodes don't receive a leader's message before a timeout, the next leader is called through a view-change.

Chained Hotstuff [**130**] can be described in two roles, the leader and voter roles. The leader role is first assigned to one node selected in a round-robin fashion, followed by the voter role assigned to every node.

Roles in Chained Hotstuff

- Voter role $\mathcal{R}_v$: Wait for message from current communication step's leader. If the SAFENODE($b^*, b^*.justify$) predicate passes, send a vote to the next leader. Decide for blocks with a three-chain, commit on blocks with a two-chain, and pre-commit

blocks with a one-chain. If no message is received from the current communication step leader, send a NEXTVIEW(m)essage to the next leader. Its r value is $2f + 1$.

- Leader role $\mathcal{R}_l$: Wait for all messages until there are $n - f$ votes and form QC with the partial signatures. Extend the highest received QC with a new leaf, containing the new input, and broadcast it. Its r value is 1.

Commitees of roles

- Committee of role $\mathcal{R}_l$: contains only one node, that communication step's leader.
- Committee of role $\mathcal{R}_v$: contains n nodes, with the same voting rules.

Role assignment

For $\mathcal{R}_l$, $assignRoles_{\mathcal{R}_l}(s)$ outputs a single node in a round robin fashion. For $\mathcal{R}_v$, $assignRoles_{\mathcal{R}_v}(s)$ outputs every node that is assigned one vote each.

The leader role is neither abundant nor concealed, preventing Chained Hotstuff's liveness against $\mathcal{A}_{MCAB}$ by Theorem 3. It is easy to see that liveness is not guaranteed against $\mathcal{A}_{MCAB}$ in streamlined BFT protocols with single leaders, for example Tendermint [**29**], Damysus [**49**], and Jolteon [**65**].

COROLLARY 3 (Security of Chained Hotstuff). *Chained Hotstuff does not achieve t-liveness against $\mathcal{A}_{MCAB}$ with $f + c \leq n/3$ for f adaptive Byzantine faults, c mobile crash faults, and n total nodes at any time.*

PROOF. The role $\mathcal{R}_l$ is not abundant since $|\mathcal{R}_l| = 1 \leq c + 1$. In addition, $\mathcal{R}_l$ is not concealed since, with the public round-robin schedule, $\mathcal{A}_{MCAB}$ can win the protected game with probability 1 by choosing to crash the next publicly known leader. By Lemma 1, Chained Hotstuff does not achieve t-liveness against $\mathcal{A}_{MCAB}$. □

3.6.4 Adding abundance: Abundant Chained Hotstuff

By running n Chained Hotstuff instances in parallel whose round-robin leader schedules don't overlap, we obtain the *Abundant Chained Hotstuff* protocol where every role is abundant, live against $\mathcal{A}_{MCAB}$. This modified protocol does not necessarily perform better practically and simply illustrates the use of abundance to maintain security against $\mathcal{A}_{MCAB}$.

We use Hotstuff-2's [**92**] improvements where only three consecutive honest leaders are required to finalize blocks, while maintaining optimistic responsiveness. To handle conflicts between instances, each instance's blocks of the same height are combined according to the instance number. This necessitates a finalized block in an instance to wait for every other block of the same height from other instances. To maintain progress at each instance in spite of faulty leaders, honest leaders extend every instance's highest QC, instead of only its own QC.

Roles in Abundant Chained Hotstuff:

- Voter role $\mathcal{R}_v$: Wait for proposals from the n instances at communication step s or after waiting for Δ. For each i, if the SAFENODE$((b_i^*, b_i^*.justify))$ predicate passes, send a vote to instance i's next leader. If no message is received or the SAFENODE$((b_i^*, b_i^*.justify)$ predicate fails, send a nextView message to instance i's next leader. Once every instance's block for the same height have a two-chain, decide for these blocks, and commit on blocks with a one-chain. There are n messages in total, either a vote or a nextView, which are sent once n messages are received, or after timeout. The r value of $\mathcal{R}_v$ is $2(f + c) + 1$.
- Leader role $\mathcal{R}_l$: Wait for all messages until there are $2(f + c) + 1$ votes and form QC with the partial signatures. Wait for up to a timeout Δ to receive the highest QC for every instance, extend them with a new leaf, containing the new input, and broadcast it. The r value of $\mathcal{R}_l$ is $f + 2c + 1$ to obtain 2 consecutive honest leaders on one instance.

Committees of roles

- Committee of role $\mathcal{R}_l$: contains n nodes, that communication step's leaders.
- Committee of role $\mathcal{R}_v$: contains n nodes.

Role assignment

For $\mathcal{R}_l$, $assignRoles_{\mathcal{R}_l}(s)$ assigns an instance to each node, rotating in round robin. For $\mathcal{R}_v$, $assignRoles_{\mathcal{R}_v}(s)$ assigns every node to vote for every proposals.

To show that Abundant Chained Hotstuff achieves persistence, we first show that Abundant Chained Hotstuff has persistence against $\mathcal{A}_{AB}$. We analyze all cases for two conflicting decisions v and w and show that each case leads to a contradiction. We then use Theorem 1 to show persistence against $\mathcal{A}_{MCAB}$.

COROLLARY 4 (Persistence of Abundant Chained Hotstuff). *Abundant Chained Hotstuff achieves persistence against $\mathcal{A}_{MCAB}$ with $n > 3(f + c)$ nodes for f adaptive Byzantine faults and c mobile crash faults at any time.*

PROOF. We first show that Abundant Chained Hotstuff has persistence against $\mathcal{A}_{AB}$.

Proof by contradiction: Assume two conflicting proposals v and w are delivered at communication steps s_v and s_w respectively. We analyze the three cases for this assumption, each leading to a contradiction.

- Case 1: an individual Chained Hotstuff instance in the protocol delivered v and w. This is a contradiction as two-chain Chained Hotstuff is safe against $\mathcal{A}_{AB}$ as shown in the original work [**130**], and can't deliver v and w.
- Case 2: v and w are delivered by separate Chained Hotstuff instances, and $s_v = s_w$. Each instance's proposal has received a two-chain, at which point every honest node will discard conflicting proposals. Only v or w is therefore delivered, a contradiction.
- Case 3: v and w are delivered by separate Chained Hotstuff instances and, w.l.o.g, $s_v < s_w$. To deliver w, w requires at least $2f + 1$ votes in one step. However, to reach $2f + 1$ votes, at least one honest node has voted for both v and w, a contradiction.

By Theorem 1, persistence is maintained against $\mathcal{A}_{MCAB}$. □

For liveness, we use the liveness against of $\mathcal{A}_{AB}$ of Hotstuff-2 [**92**] to show that individual Abundant Chained Hotstuff instances have liveness. We then show that every role is abundant, implying liveness against $\mathcal{A}_{MCAB}$ by Theorem 3.

COROLLARY 5 (Liveness of Abundant Chained Hotstuff). *Abundant Chained Hotstuff achieves liveness against $\mathcal{A}_{MCAB}$ with $n > 3(f + c)$ nodes for f adaptive Byzantine faults and c mobile crash faults in a partially synchronous network.*

PROOF. Hotstuff-2 has liveness after GST against $\mathcal{A}_{AB}$ as shown in the original paper [**92**]. For its chained version, it is known that liveness additionally requires three consecutive honest leaders to commit a proposal thanks to previous work on Chained Hotstuff [**69**, **130**]. After GST when $n > 3f$ with round-robin leaders, three consecutive honest leaders are eventually obtained and Chained Hotstuff-2 also has liveness after GST against $\mathcal{A}_{AB}$. In Abundant Chained Hotstuff, each instance has round-robin leaders and obtains three consecutive honest leaders after GST against $\mathcal{A}_{AB}$.

To show that Abundant Chained Hotstuff maintains liveness against $\mathcal{A}_{MCAB}$, we show that both roles are abundant. $\mathcal{R}_v$ is abundant since $|\mathcal{R}_v| = n \geq\ c + 2f + 1$ every communication step where it is assigned. For $\mathcal{R}_l$ to be abundant, $|\mathcal{R}_l|$ must be large enough so that at least one instance is able to have three consecutive honest leaders. In three leader phases, $\mathcal{A}_{MCAB}$ is able to mobile crash the leader of $3c$ instances, while $3f$ instances may have a Byzantine leader. We therefore require $3f + 3c + 1$ leader roles assigned to distinct nodes. $\mathcal{R}_l$ is abundant since $|\mathcal{R}_l| = 3f + 3c + 1 \geq\ 3f + 3c + 1$ every communication step where it is assigned. By Theorem 3, Abundant Chained Hotstuff has persistence in $\mathcal{A}_{MCAB}$. □

3.6.5 Extensions and future directions

Generic frameworks for asynchronous fallback protocols [**65**, **88**] may be modified for the MCAB setting, such that a protocol can switch between a protocol without concealment or abundance, vulnerable to DoS attacks, and a protocol secure against $\mathcal{A}_{MCAB}$. Knowing how to efficiently switch modes involves subtleties explored for the asynchronous case, for MCAB fallback we may follow the general framework [**88**], but dedicated redesign might be needed.

For example, differentiating between simple leader failures, mobile DoS attacks, and network failures requires care.

The ACE framework [**122**] has a similar design to Abundant Hotstuff as it runs multiple instances of leader-based protocols in parallel, however it is designed to add asynchronous liveness while Abundant Hotstuff is secure in a partially synchronous network against $\mathcal{A}_{MCAB}$.

The condition of abundance or concealment may be a starting point to generalize Byzantine quorum systems' [**13**] availability property with parameters r and q. Instead of requiring the number of honest roles in a committee to be a full quorum, only r are required, and it must hold only for at least q communication steps in t, as opposed to at all steps. Links between Byzantine quorum systems [**13**] and conditions in the MCAB model could be further explored.

Optimal resilience bounds in the MCAB are still open. In a synchronous network where $n > 2f + c$ with MCAB, or the sleepy model, no known deterministic protocol exists to our knowledge. In partial synchrony, Abundant Chained Hotstuff is an example of a deterministic protocol with $n > 3(f + c)$. When the crashes are static and not mobile, $n > 3f + 2c$ has been achieved [**73**].

3.7 Conclusion

In this work, we identified and filled a gap for generic adversary models, between traditional blockchain adversaries unable to model mobile DoS, and existing models that do model mobile DoS but only under restrictive conditions. To understand the security requirements of our model, we proved that *abundance* and *concealment* are sufficient, and that at least one of them is necessary for security. Additionally, we analyzed the trade-offs between the two properties. We applied our findings to evaluate the security of Ouroboros Praos, Algorand, and Hotstuff as case studies, confirming the intuition that Ouroboros Praos and Algorand tolerate a stronger adversary.

CHAPTER 4

Constant latency dynamic DAG and finality.

4.1 Introduction

Bitcoin [**102**] introduced the use of blockchain to enable nodes in a permissionless environment to agree on an ordered set of transactions. In this setting, nodes are free to join or leave the network at any time. Consequently, Bitcoin guarantees liveness in a permissionless environment, referred to as dynamic availability. However, to ensure security, the network must be synchronous, meaning messages must be delivered within a known time bound. It follows that Bitcoin does not provide safety in the face of network partitions, where the network becomes asynchronous and message delivery time is unknown. A further known limitation of Bitcoin and its derivatives is performance: latency ranges from minutes to hours, and throughput is fewer than 10 transactions per second.

To remedy the limited performance of Bitcoin's blockchain consensus, traditional Byzantine Fault Tolerant State Machine Replication (BFT-SMR) techniques have been considered for the blockchain setting. Some BFT-SMR protocols are able to remain secure even under network partitions and asynchronous communication networks [**29**, **36**, **97**, **130**]. However, a trade-off with BFT-SMR is that the number of active nodes in the system must be known, limiting dynamic availability. To address this challenge, the sleepy model [**107**] was introduced, formalizing the concept of permissionless networks. Recent advances in BFT within the sleepy model have achieved constant expected latency [**91**, **100**] by regularly estimating for the current participation level.

While, ideally, both dynamic availability (liveness) and security during network partitions (safety) are achieved within a single protocol, the CAP theorem demonstrates this to be impossible [**68**]. Consequently, two distinct approaches have emerged: one prioritizes liveness with dynamic availability, and the other focuses on asynchronous safety.

Originally, both classes of protocols include transactions in a linear sequence of batches or blocks. However, utilizing a Directed Acyclic Graph (DAG) structure, blocks are processed concurrently which has significantly increased throughput for both approaches. This innovation allows throughput to scale with network capacity [**22**, **46**, **121**]. Using provable dispersal for block propagation has achieved similar throughput improvements as the DAG structure [**85**,**89**,**129**], however, to the best of our knowledge, implementing provable dispersal in the sleepy model remains an open, orthogonal problem.

We highlight the following research gaps regarding DAG based protocols in each line of work.

DAGs when prioritizing liveness. As noted earlier, proposals for BFT in the sleepy model have achieved constant latency. Meanwhile, recent advances in DAG-based protocols have primarily focused on asynchronous and partially synchronous networks [**46**, **121**], leaving the modern use of DAGs in BFT for the sleepy model largely unexplored.

Although DAG-based proposals expanding on Bitcoin's longest chain idea exist [**22**, **118**], their latency is linear in certain security parameters. Achieving constant latency for high throughput DAG protocols within the sleepy model remains a significant open challenge.

DAGs when prioritizing safety. DAG-based protocols such as Narwhal/Tusk [**46**] and Bullshark [**121**] disseminate an unconfirmed DAG of blocks as a mempool, of which a subset is finalized. However, there are multiple approaches for the finalization process, and previous work has not provided a generic security analysis of finalization approaches in a given DAG protocol. For instance, one way to finalize Narwhal is by using a separate consensus primitive such as Hotstuff to agree on a block and all its ancestors, as presented by the authors. Alternatively, Tusk, which builds on top of Narwhal, selects leaders using a common coin but does not independently solve consensus. Similarly, Bullshark employs a round-robin leader

schedule to finalize a subset of the DAG, but the finalization mechanism in isolation is not a consensus protocol in itself.

We observe that the common thread in the two latter protocols is a finalization mechanism that leverages the DAG structure to obtain consensus in the overall system. A key advantage of this approach is that it avoids the need to layer a full BFT protocol on top of the DAG dissemination mechanism. However, there is no composable treatment of DAG finalization in the literature beyond constructions specific to monolithic DAGs.

Contributions

In this work we address the research gaps previously highlighted, and leverage our findings to propose a flexible protocol simultaneously catering to clients favoring liveness and clients favoring safety.

We introduce the first constant latency DAG based protocol in the sleepy model. Our protocol can commit a set of DAG blocks in 3Δ latency, where Δ represents the synchronous network's message delivery bound. This result not only advances upon existing constant latency sleepy model BFT, but also improves on existing DAG protocols whose latency linearly scales with Δ. We provide a summary comparison of our protocol with other dynamic availability protocols in Table 4.1. Additionally, we experimentally show the improved performance of our DAG protocol compared to Mahlki et al. constant latency protocol. We can observe that our DAG protocol indeed achieves drastic throughput improvements over the baseline due to its parallel block proposals, and that it has lower expected latency.

Secondly, we introduce a generic primitive for DAG finalization, named *graded common prefix*, and an efficient construction. Graded common prefix allows a set of nodes to quickly agree on the subset of a DAG common to all honest nodes. Crucially, our primitive is lighter than standard consensus, enabling simpler protocols. For example, our construction runs in only 2 communication steps, compared to the 4 expected steps per decision required by low latency partially synchronous BFT [**29**, **36**, **124**]. It is also resilient to crashes within the supermajority bound, unlike many existing leader-based protocols for partially synchronous

TABLE 4.1. Comparison of optimal resilience SMR in the sleepy model. k is the security level and a is the fraction of awake nodes.

	Avoid single block bottleneck*	Good case latency
Bitcoin [**102**]	✗	$O(k\Delta/a)$
Ouroboros/ Snow White [**20**, **44**]	✗	$O(k\Delta/a)$
Pass et al. [**107**]	✗	$O(k\Delta/a)$
Goyal et al. [**72**]	✗	$O(k\Delta)$
Prism [**22**]	✓	$O(\Delta/a)$
Garay et al. [**62**]	✓	$O(\Delta/a)$
Momose et al. [**100**]	✗	16Δ
Mahlki et al. [**91**]	✗	4Δ
(This work)	✓	3Δ

* Protocols with a single block bottleneck commit blocks sequentially without provable dispersal or DAG techniques to increase throughput.

networks [**29**, **36**, **124**, **130**]. We experimentally observe the resulting improved latency of our construction over a leader-based BFT, Hotstuff [**130**], with and without crashes. Graded common prefix can also be defined generically with sets of values instead of block-DAGs, which may be of independent interest for distributed protocols.

Thirdly, we propose a flexible protocol that consists of two sub-protocols running concurrently, namely a dynamically available sub-protocol and a partially synchronous finality sub-protocol. Clients choose which output (of the sub-protocols) they consider acceptable according to their preference (security v.s. liveness). To accomplish this, we adapt the Ebb-and-Flow framework [**104**], which, roughly speaking, considers the sleepy model with a *Global Awake Time* (GAT), the time after which all honest nodes participate actively. GAT guarantees that the finality sub-protocol eventually progresses, as dynamic availability is not required after GAT. To maintain consistency, the output of the dynamically available sub-protocol is used as input to the finality sub-protocol. We show that graded common prefix is sufficient for finality after GAT, avoiding the need to perform standard consensus twice as in existing schemes. Complementing our flexible protocol, we further relax the GAT assumption by introducing *Quorum Awake Time* (QAT). Unlike GAT, which assumes that all honest nodes are awake after this period, QAT requires only that a sufficient number of honest nodes are awake. To quantify the number of awake nodes after QAT, we leverage the recently introduced Mobile

Crash Adaptive Byzantine (MCAB) model [**111**] that enables a fine-grained treatment of the upper bound on crashed/sleepy nodes for a protocol.

The rest of this paper is organized as follows. Section 4.2 introduces the system model and preliminaries used in this work. Section 4.3 presents our dynamically available DAG protocol and the formal definitions and properties it achieves. Section 4.4 introduces the graded common prefix primitive and a construction for it. Section 4.5 synthesizes our results into the flexible protocol described previously, then presents QAT, the generalization of GAT. Section 4.6 contains the formal security proofs of our theorems and lemmas. Section 4.7 provides the experimental evaluation of our DAG protocol. The conclusion summarizes our findings.

In summary:

- We propose a constant latency dynamically available DAG BFT protocol with 3Δ latency. Its high throughput is demonstrated experimentally.
- We introduce *graded common prefix* (GCP), a primitive that allows a set of nodes to agree on the subset of a DAG in common to honest nodes. Additionally, we provide a construction of GCP that runs in 2 communication steps. Its low latency and resilience to crashes is shown experimentally.
- We show that GCP is sufficient as a finality sub-protocol combined with a dynamically available sub-protocol, to allow clients to prioritize dynamically available liveness or asynchronous safety. In addition, we introduce and discuss a generalization of *Global Awake Time* of Ebb-and-Flow [**104**], the time after which honest nodes are assumed to be awake to allow asynchronous safety. Our generalization, *Quorum Awake Time*, relaxes the GAT assumption by leaving the possibility of a parametrized bound on the number of sleepy nodes.

4.2 System model and preliminaries

4.2.1 System model

We consider a set $\mathcal{N}$ of nodes, each equipped with a public key and a unique identifier. This includes the Proof-of-Stake setting where the smallest discrete amount of stake is represented by one node. Time advances in discrete slots and, for simplicity, nodes have access to synchronized clocks. Our results can include bounded clock drifts using a round transformation technique [**100**] at the cost of small added latency.

Node corruptions. Nodes may be controlled by an *adversary*, selected during the execution of a protocol. Nodes controlled by the adversary are considered Byzantine, and may behave arbitrarily. When a node is Byzantine at slot t, it stays as Byzantine for all slots $t' > t$. The remaining nodes are *honest*, and behave according to protocol specifications.

Network communication. Nodes communicate with each other through authenticated messages, within the limitations set by one of the following network assumptions.

- *Synchronous network*. All messages sent between honest nodes are delivered within a known number of time slots Δ. The order in which they arrive can be determined by the adversary.
- *Partially synchronous network*. Message delivery is determined by the adversary, as long as all messages sent between honest nodes are eventually delivered. Messages between honest nodes cannot be dropped or modified. After a *Global Stabilisation time* (GST), the network behaves as a synchronous network. GST is decided by the adversary, and unknown to honest nodes.

Sleepy model. The adversary may cause honest nodes to become *sleepy* at any time slot, during which the protocol is not executed and messages are delayed until the next slot where the node is *awake* again. Awake nodes behave according to protocol specifications. The number of awake nodes at any time slot t is denoted n_t. The unknown time after which all

honest nodes in $\mathcal{N}$ are awake is called *Global Awake time* (GAT). GAT is decided by the adversary, and unknown to honest nodes.

Parameterized dynamic participation. We use the extended sleepy model [**91**] that allows corrupted nodes' participation to be dynamic. We define $f(T_f, T_b, t)$ as the number of corrupted nodes around time t for a time T_f forward and a time T_b backwards. For a system model's fraction β of corrupted nodes, we have $f(T_f, T_b, t) < \beta \cdot n_t$. It has been shown in existing work that $T_f = \infty$ and $T_b = O(\Delta)$ is necessary for consensus under dynamic participation without PoW [**91**]. We therefore consider $T_f = \infty$ and $T_b = O(\Delta)$.

Two system models. As it is impossible to achieve dynamic availability and asynchronous finality under the same protocol [**68**], we consider two different system models to allow clients to prioritize liveness or safety, as proposed by the Ebb-and-Flow framework [**104**]. A system model is defined by an adversary $\mathcal{A}$, an environment $\mathcal{Z}$, and a fraction β of corrupted nodes.

System model $(\mathcal{A}_1(\beta), \mathcal{Z}_1)$: In this system model, the network is partially synchronous and there exist a bounded GAT and a bounded GST.

System model $(\mathcal{A}_2(\beta), \mathcal{Z}_2)$: In this system model, the network is always synchronous (GST=0) and GAT can be unbounded (GAT $\rightarrow \infty$).

4.2.2 Preliminaries

DEFINITION 13 (State machine replication protocol.). *A state machine replication protocol **P** takes transactions as input, and outputs a ledger LOG. It is secure if it satisfies the following properties for a polynomial function T_{fin} of security parameter κ.*

- Safety*: if at time t an awake honest node reports a ledger LOG, then for every honest node that reports a ledger LOG' after time $t' \leq t$, $LOG \preceq LOG'$ or $LOG' \preceq LOG$ where $\preceq$ denotes a prefix relation.*
- Liveness*: a transaction given as input to all honest nodes continuously from time t to time $t + T_{fin}$ will be reported in an honest node's output LOG after time $t + T_{fin}$.*

DEFINITION 14. *A flexible protocol is a pair of SMR protocols $(\boldsymbol{P}_1, \boldsymbol{P}_2)$ taking the same transactions as inputs and outputting the ledgers LOG_1 and LOG_2 respectively.*

Negligible function. A negligible function refers to a function $negl(x)$ where for every $y \in \mathbb{N}$, there exists an $z \in \mathbb{N}$ such that $negl(x) < 1/x^y$ for all $x \geq z$.

DEFINITION 15 (Ebb-and-flow protocol. [**104**]). *A (β_1, β_2)-secure Ebb-and-flow protocol $\boldsymbol{P}$ is a flexible protocol $(\boldsymbol{P}_{da}, \boldsymbol{P}_{fin})$ with the same input transactions and outputs an available ledger LOG_{da} and a finalized ledger LOG_{fin}, such that for security parameter κ:*

- *Finality: Under $(\mathcal{A}_1(\beta_1), \mathcal{Z}_1)$, LOG_{fin} is safe at all times, and there exists a constant C such that LOG_{fin} is live after $C(\max\{GST, GAT\} + \kappa)$ except with probability negligible in κ.*
- *Dynamic availability: Under $(\mathcal{A}_2(\beta_2), \mathcal{Z}_2)$, LOG_{da} is a secure SMR output except with probability negligible in κ.*
- *Prefix: For any honest node i and time $t > 0$, $LOG^t_{fin,i}$ is a prefix of $LOG^t_{da,i}$.*

4.3 Dynamically available DAG

This section presents the first constant latency dynamically available DAG-based protocol in the sleepy model, introducing the possibility of high throughput in sleepy model BFT, while reducing expected latency to 3Δ. Our construction is secure under $(\mathcal{A}_2(\beta_2), \mathcal{Z}_2)$ where $\beta_2 < 1/2$. We first summarize Synchronous Hotstuff [**9**] and the intuition behind its design that our protocol is based on.

Protocol flow. A leader proposes a block and honest nodes vote for it if its valid. Honest nodes can then commit after waiting 2Δ, provided no conflicting vote was received. **Security intuition.** Since a synchronous network is considered, it is guaranteed that if an honest node commits 2Δ after voting, all honest node receive the proposal and there is no conflicting commitment.

Considering the sleepy model. Waiting 2Δ for conflicts instead of collecting a fixed number of votes, a quorum, to advance is an especially interesting technique in the sleepy model, where fixed quorums are not useful. However, in Synchronous Hotstuff, potentially faulty leaders must still be dealt with, in a process called view-change that uses fixed quorums.

From single leader to DAG proposals. For our protocol, we leverage Synchronous Hotstuff's technique of waiting 2Δ after voting to detect conflicts in dynamic participation and guarantee consistency. By having every node propose a block to the block-DAG, we are able to avoid the fixed quorum based view change of Synchronous Hotstuff, guaranteeing progress in the sleepy model.

Latency benefit. Since Synchronous Hotstuff's 2Δ wait mechanism does not need explicit quorums, time-shifted quorums [**91**, **100**], obtained by round-trip communication (2Δ), are not necessary. Therefore, we are able to reduce expected latency to 3Δ, from 4δ in the state-of-the-art [**91**] .

4.3.1 Block-DAGs and structured dissemination

We define a Directed Acyclic Graphs of blocks (block-DAG) which we use as a foundation for our DAG based protocols. Formally defining block-DAGs allows us to reason about a generic DAG protocol without specifying a specific algorithm. Block-DAGs can be seen as the generalization of the blockchain data structure.

DEFINITION 16. *block-DAG. A block-DAG $D = \{\mathcal{E}, \mathcal{B}\}$ comprises a set $\mathcal{E}$ of directed edges and a set $\mathcal{B}$ of vertices, as follows:*

- *A directed edge $(B \to B') \in \mathcal{E}$ represents a transitive relation of B referencing B', where $B, B' \in \mathcal{B}$. B' is a parent of B, denoted $B' \in B.parents$.*
- *A vertex $B \in \mathcal{B}$ is called a block. Every block references at least one other vertex, except the genesis block $B_{genesis}$ where $B_{genesis}.parents = \varnothing$. The height of a vertex is $B.height = h + 1$, where $h = \max(B.parents.height)$ and $B_{genesis}.height = 0$.*

B is a tip if $\nexists B' \in \mathcal{B}$ s.t. $B \in B'.parents$. Each block B has an author $B.author$ and $B_{genesis}.author = \varnothing$.

- *Extension: A block-DAG $D' = \{\mathcal{E}', \mathcal{B}'\}$ is an extension of D, denoted $D \subseteq D'$, if $\mathcal{E} \subseteq \mathcal{E}'$ and $\mathcal{B} \subseteq \mathcal{B}'$.*
- *Height: The height h of D is $\max(\{B.height | B \in \mathcal{B}\})$.*
- *Conflict: D and D' are conflict if $(D \nsubseteq D') \wedge (D' \nsubseteq D)$, denoted $D \nsim D'$. Otherwise, we use $D \sim D'$.*

A block B, contains referenced blocks $B.parents$ which recursively reference blocks until $B_{genesis}$, representing a block-DAG whose highest height block is B. We can therefore uniquely represents a sub-graph of D with height h by the set of identifiers for blocks in D of height h.

We define properties for structured dissemination, guaranteeing that a distributed set of nodes maintaining a block-DAG is able to obtain a consistent, growing view of the block-DAG.

DEFINITION 17 (Structured dissemination.). *Structured dissemination with parameters (k, τ, t, μ) takes transactions as input and outputs a block-DAG. A structured dissemination protocol is secure if it satisfies the following properties:*

- *δ-Consistency: If a block B of height h is considered consistent (called confirmed) in an honest node's block-DAG at time t, B and its ancestors are confirmed in any honest node i's block-DAG D_i at time $t + \delta$, and no honest node confirms a block of height h after $t + \delta$.*
- *τ, t-DAG growth: any honest node's view of the block-DAG at time $t' + t$ must include $\tau \cdot t$ additional blocks proposed by honest nodes compared to its view at time t'.*
- *μ-DAG quality: any honest node's view of the block-DAG includes at least a fraction μ of blocks proposed by honest nodes.*
- *External Integrity: If a block B is consistent, $condition(B) = 1$ holds for the protocol-specific external integrity condition.*

Obtaining total order. With δ-consistency among honest nodes, confirmed blocks can be ordered by height. In the blockchain special case, an ordered ledger is obtained trivially since we have only one confirmed block per height. In a block-DAG, a height h may have multiple confirmed blocks. However, with δ-consistency, honest nodes confirm the same set $\mathcal{B}$ of blocks for height h. Once blocks in $\mathcal{B}$ are confirmed, a simple rule is therefore already enough to totally order them, for example ordering based on block proposers' identifiers.

Designing more intricate block-DAG ordering rules that may achieve application-specific fairness requirements is orthogonal to this work.

Example in the literature. To illustrate our block-DAG and structured dissemination definitions in the context of existing DAG based protcols, we show that the popular Narwhal [**46**] DAG mempool layer is a block-DAGs that satisfies the properties of structured dissemination under $(\mathcal{A}_2(\alpha), \mathcal{Z}_2)$ after GAT in Lemma 3. Recall that $(\mathcal{A}_2(\alpha), \mathcal{Z}_2)$ after GAT has a synchronous network and honest nodes are awake, enabling Narwhal to achieve structured dissemination properties while not guaranteeing total order in its original, asynchronous network system model. To translate Narwhal's nomenclature into a block-DAG, consider the following: a block is considered confirmed when it is *certified*, a block b extends a block b' if it includes its certificate, the height of a block is the round in belongs to.

LEMMA 3. *Narwhal run by n nodes is a structured dissemination of a block-DAG with $\Delta-$consistency, $\frac{2n}{9\Delta}, 3\Delta$-DAG growth, and $1/2$-DAG quality under $(\mathcal{A}_2(\alpha), \mathcal{Z}_2)$ after GAT where $\beta_2 < 1/3$.*

Proof. Δ-Consistency: We proceed by contradiction. Assume an honest node u considers a block b with digest d consistent, but another honest node v does not. Since u considers b confirmed at time t, it is certified and the *write(d,b* operation has completed. By the block-availability property of Narwhal and the synchronous network assumption, *read(d)* completes by time $t + \Delta$. By the integrity property of Narwhal, *d* outputs the same block b for all honest nodes, completing our contradiction.

Assume a block b of round r and height h is consistent at time t in an honest node u's block-DAG, but another honest node v confirms a block $b' \neq b$ of round r and height h after

$t + \Delta$. Since the network is synchronous, b' is received after b is certified, implying that honest blocks of round r are also certified. This further implies that honest nodes are at least in round $r + 1$ since they certified $2f + 1$ honest blocks of round r. b' is therefore not valid for honest nodes due to the round number, and unable to obtain a quorum certificate, a contradiction.

$\frac{2n}{9\Delta}, 3\Delta$-*DAG growth:* We proceed by induction. For the first round of the protocol, at least $2n/3$ honest node propose a block certified within 3Δ time. For a round $r + 1$, every honest node has received at least $2n/3$ blocks from honest nodes at round r and is able propose a block delivered within Δ time. We therefore have $\frac{2n}{3}$ blocks certified in 3Δ time, and $\tau = \frac{2n}{9\Delta}$, completing the induction.

$1/2$-*DAG quality:* $1/2$-DAG quality directly follows from Narwhal's $1/2$-chain quality property.

External Integrity: External integrity for $condition(B)$ can be trivially satisfied by adding $condition(B) = 1$ as a requirement for honest nodes to certify blocks. If $condition(B) \neq 1$ for a block B, B can not obtain a certificate from at least $2f + 1$ nodes' signatures.

4.3.2 Overview

We outline the main components of our protocol and provide brief intuition behind its components.

Propose. Every Δ slots, nodes propose a block of transactions sent by clients.

Vote. When proposing a block of height $h + 1$, nodes include as references all received blocks with height h, and start their confirmation timer for the set of referenced blocks at height h.

Confirm. A block of height h proposed at time t is considered confirmed 2Δ slots after the confirmation timer for height h has started if no conflicts are detected for blocks of height h. The confirmation timer starts at time $t + \Delta$ when proposing a block of height $h + 1$, such that, by the time $t + 3\Delta$, it is confirmed and received by all honest nodes.

Order blocks. As mentioned in the previous section, confirmed blocks of the same height h can be deterministically ordered by their proposer's identifiers.

Byzantine agreement for conflicts. Consider a Byzantine node's conflicting block proposals B and B'. Simply discarding both is not possible as some honest nodes may consider B confirmed as illustrated in Figure 4.1b. If both are discarded by node u_1, node u_i may confirm the blue block.

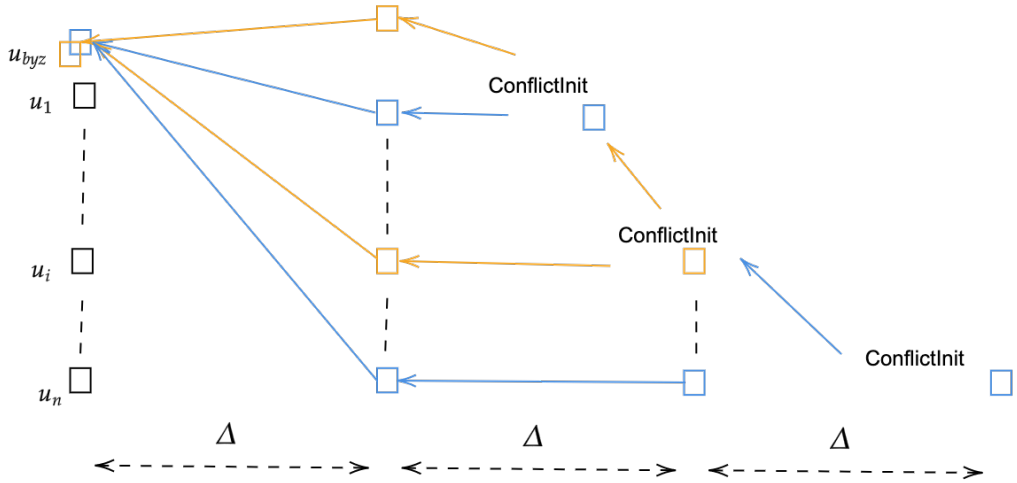
When an honest node broadcasts a conflict message including B and B', v is provably Byzantine. To handle equivocating proposals by a Byzantine node $B.author$, nodes concurrently run a generic Byzantine agreement sub-protocol BYZAGREEMENT as in Definition 18 with the first received block B as input. $B.author$ serves as an identifier for the Byzantine agreement protocol instance. Blocks of a height h are only confirmed once all potential instances of BYZAGREEMENT for height h are completed.

DEFINITION 18 (Byzantine Agreement (BA).). *In a Byzantine agreement protocol, each node i has an input value v_i, and outputs a value v. A Byzantine agreement protocol satisfies the following properties:*

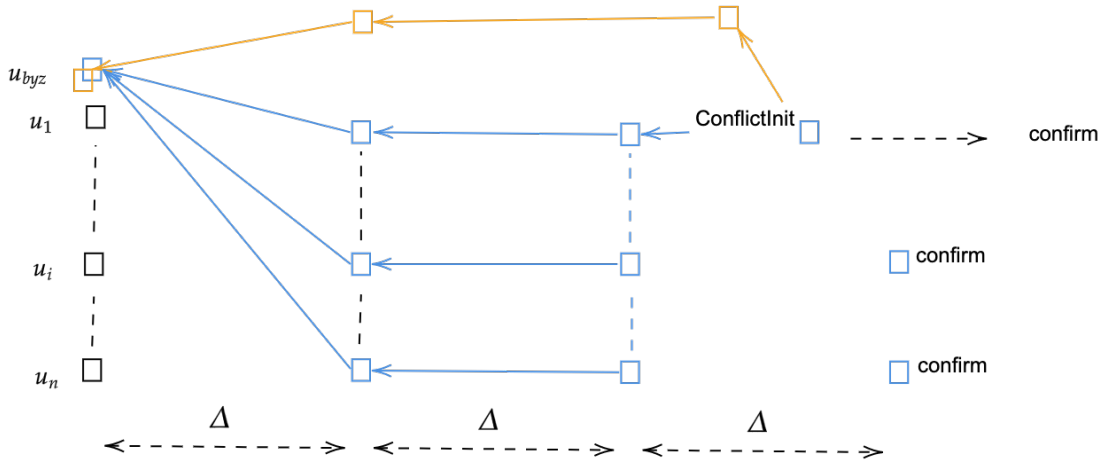
- *Termination: Every honest node outputs a value and terminates.*
- *Validity: If all honest nodes propose the same value v, all honest nodes output v.*
- *Agreement: If an honest node outputs v then all honest nodes output v.*

BYZAGREEMENT can be instantiated with existing sleepy model protocols from the literature [**45**, **91**, **100**], and benefits from future advances in Byzantine agreement in the sleepy model. Additional timers may be required for the specific BYZAGREEMENT implementation.

Conflict cases. To understand why all conflicts are resolved, consider the two cases depicted in Figure 4.1. In Figure 4.1a, the first honest node u_1 receiving the conflict does so before 2Δ, guaranteeing that all honest nodes receive the conflict before honest nodes confirm a proposal. Some nodes may have received the orange block first while others may have received the blue block first. BYZAGREEMENT then guarantees agreement for the blue or orange block.



(A) Conflict is first received before 2Δ.



(B) Conflict is first received after 2Δ.

FIGURE 4.1. Two cases of conflicting proposals.

In Figure 4.1b, the first honest node u_1 receiving the conflict does so after 2Δ. Honest nodes have seen blue first when entering BYZAGREEMENT, and some may have confirmed blue already. As all honest nodes input blue to BYZAGREEMENT, blue is the output.

Handling late blocks A node u may also receive a block B of height h at time $t' \geq t + \Delta$ from a Byzantine node where t is the time that u proposes a block of height h. Here, u

knows that $B.author$ is Byzantine, but unlike in conflicts, it is not straightforward to prove that u hasn't received h before $t + 2\Delta$. Node u broadcasts an UNRECEIVED message for B including all received blocks of height $h + 1$. Upon receiving an UNRECEIVED message for B, nodes forward the message and discard B if it is referenced by a minority of known blocks for height $h + 1$.

A node u checks all blocks of height $h + 1$ received by time $t + 2\Delta$ and if none reference B, broadcasts the set $\mathcal{B}_{h+1}$ of all received blocks of height $h + 1$ received by time $t + 3\Delta$ in a UNRECEIVED message. Upon receiving an UNRECEIVED message, a node rebroadcasts the UNRECEIVED message and discards B if it is referenced by a minority of blocks received for height $h + 1$ when considering $\mathcal{B}_{h+1}$ in addition to their received blocks.

A Byzantine node can not force honest blocks of height h to be discarded as they are referenced by all honest nodes at height $h + 1$, the majority. A Byzantine block can achieve a majority only if at least one honest node references it in a block of height $h + 1$, guaranteeing that it is shared among all honest nodes before $t + 2\Delta$.

Latency under Byzantine behavior Byzantine nodes can increase latency through malicious behavior. However, equivocations are easy to detect with the conflict resolution messages, and can be punished in a deployed system.

In terms of added latency, the latency Δ_{BA} of the BYZAGREEMENT instantiation would be added for a latency of $3\Delta + \Delta_{BA}$. The best expected latencies for Byzantine Agreement in the sleepy model in the literature at the time of writing, to our knowledge, are 4Δ expected latency [**91**].

In the worst case, it is known that any consensus protocol requires at least O(f) rounds of communication under f faults [**12**]. Similarly, our protocol also adheres to this lower bound.

4.3.3 Full protocol

Protocol flow We provide an overview of $\mathbf{P}_{sd}$ from Algorithm 1, illustrated in Figure 4.2. The local view of the full block-DAG D and confirmed block-DAG D_{con} are initialized

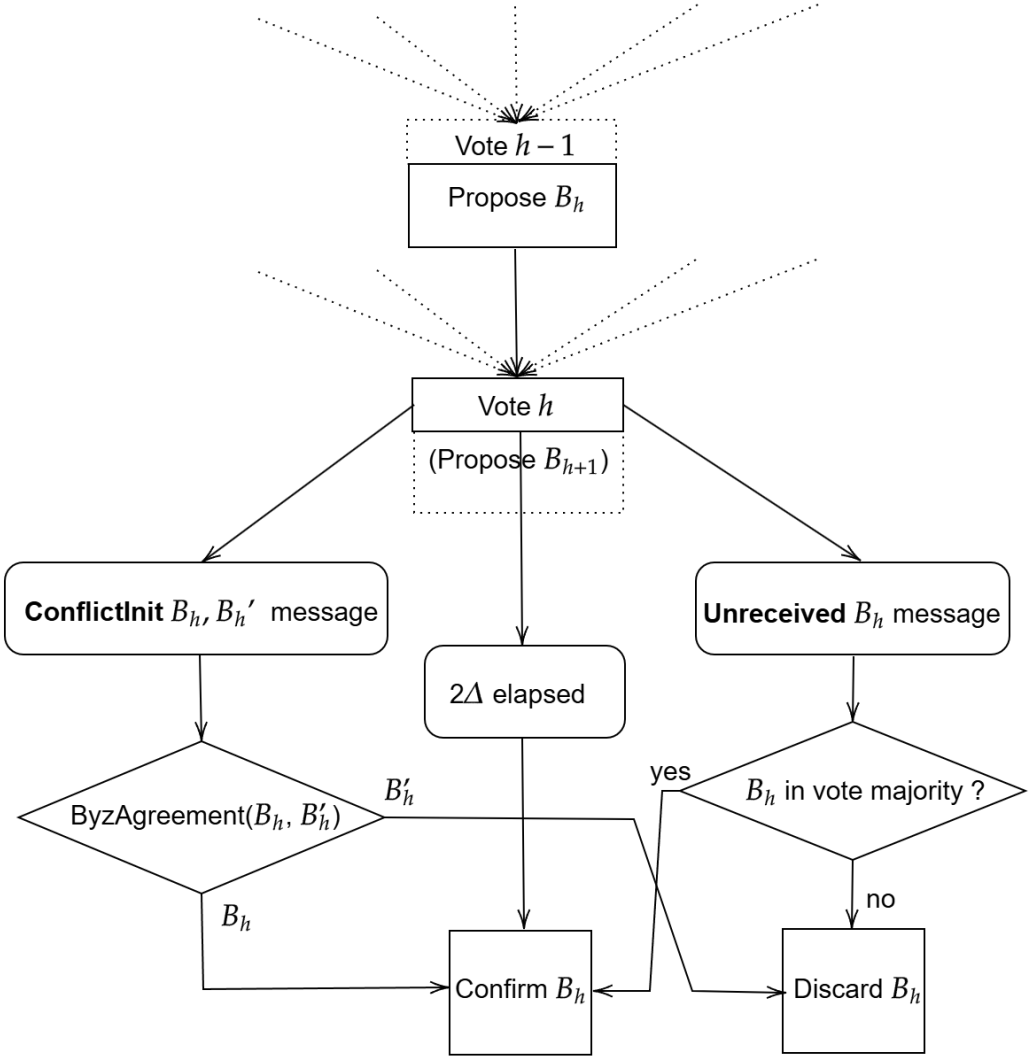


FIGURE 4.2. Sequence diagram for the confirmation process of a block B_h at height h in our dynamically available DAG protocol. The Propose message for B_h includes the Vote message for blocks of height $h-1$. The Vote message for height h includes a next proposal B_{h+1} for height $h+1$.

by the genesis block (Line 1-3). Each time slot, network messages are processed. Newly received blocks are processed according to Lines 25-40 described below and conflict messages according to Lines 41-61 describe below (Line 5-8).

Every Δ time slots, RESETBLOCKTIMER(T) resets the block timer, which can be accessed by BLOCKTIMER(). Upon RESETBLOCKTIMER(T), nodes broadcast a block of height k

extending all received blocks of height $k-1$ (Line 9-13). The timer for consistency is started, and the height to confirm is saved (Line 14-15). Nodes consider a block B of height h proposed by node u consistent Δ time after referencing B in its own block proposal at height $h+1$, unless a CONFLICTINIT message is received. If a CONFLICTINIT message is received for a block of height h, the consistency timer is reset to allow time to resolve conflicts (Line 17-21).

New received blocks are processed in SDPROCESSNEWBLOCKS. If a block $B' \neq B$ of height h proposed by u is received before receiving B, they are broadcast in a CONFLICTINIT message including B' and B, and enter a BYZAGREEMENT(B') instance with input B' as B' was received first (Line 26-29). Otherwise a valid block whose height is at least the consistency height is added to the block-DAG (Line 30-33). A block B is valid if $condition(B) = 1$ holds for the application-specific external integrity condition $condition(B)$. If a block of height h is received 2Δ time after proposing a block of height h, an UNRECEIVED message is broadcast including B and all blocks of height $h+1$.

CONFLICTINIT and UNRECEIVED messages received at slot t are processed in PROCESSCONFLICTMESSAGES For a CONFLICTINIT message containing B and B', nodes forward CONFLICTINIT and enter the black-box BYZAGREEMENT protocol (Line 42-45).

For an UNRECEIVED message containing the block B and the set $\mathcal{B}_{B.height+1}$ of blocks, a node adds blocks in $\mathcal{B}_{B.height+1}$ to its set $\mathcal{B}'_{B.height+1}$ of blocks of height $B.height+1$ and forwards the UNRECEIVED message with $\mathcal{B}'_{B.height+1}$ replacing $\mathcal{B}_{B.height+1}$ (Line 48-52). If B is referenced in a minority of blocks in $\mathcal{B}'_{B.height+1}$, B is discarded from the block-DAG D (Line 53-54).

If a BYZAGREEMENT outputs a block B_{out}, all other blocks of height $B_{out}.height$ by $B_{out}.author$ are discarded from the block-DAG D (Line 57-60).

Algorithm 1 Dynamically available structured dissemination protocol $\mathbf{P}_{sd}$

1: **procedure** MAIN()
2: $D, D_{con} \leftarrow B_{genesis}$ // initialize block-DAG D and confirmed block-DAG D_{con} with genesis block
3: $h_{con} \leftarrow 0$ // initialize to-confirm block height
4: **for** time slot $t \in \mathbb{N}$ **do**

5: PROCESSINCOMINGNETWORKMESSAGES()
6: $\mathcal{B}_{new} \leftarrow$ GETLATESTBLOCKS()
7: $D \leftarrow$ SDPROCESSNEWBLOCKS$(D, \mathcal{B}_{new}, height_{con})$ // see line 25
8: $D \leftarrow$ PROCESSCONFLICTMESSAGESSLOT(t, D) // see line 36
9: **if** BLOCKTIMER$(t) = 0$ **then**
10: $B \leftarrow$ CREATEBLOCK$(clienttransactions)$
11: $B.parents \leftarrow$ { All blocks of height $D.height$ }
12: BROADCAST$(< Propose, B >)$
13: RESETBLOCKTIMER(t)
14: $h_{con} \leftarrow B.height - 1$
15: STARTCONSISTENCYTIMER(t, h_{con}) // start timer for height h_{con}
16: **end if**
17: **if** (CONSISTENCYTIMER$(t, \mathcal{B}_{con}) = 0) \wedge (\exists$BYZAGREE with blocks in $\mathcal{B}_{con}$ **then** // check for ongoing conflict resolution instances
18: RESETCONSISTENCYTIMER$(t, \mathcal{B}_{con})$
19: **else if** CONSISTENCYTIMER$(t, \mathcal{B}_{con}) = 0$ **then**
20: $D_{con} \leftarrow D_{con} \cup \mathcal{B}_{con}$
21: **end if**
22: **end for**
23: **end procedure**

24: **procedure** SDPROCESSNEWBLOCKS$(D, \mathcal{B}_{new})$
25: **for** $B \in \mathcal{B}_{new}$ **do**
26: **if** $\exists B' \in D$ s.t. $B'.author = B.author \wedge B'.height = B.height$ **then**
27: // block B' received first
28: BROADCAST$(<$ CONFLICTINIT$, B', B >)$
29: BYZAGREEMENT(B') // input B' to BA
30: **else if** $condition(B) = 1$ **and** $B.height \geq D.height$ **then**
31: // check external integrity condition on B
32: // honest blocks of lower height are indirectly included by newer blocks
33: $D \leftarrow D \cup B$
34: **else if** $\Delta_{B.height} > 2\Delta$ where $\Delta_{B.height}$ denotes the time since block B of height $B.height$ was proposed **then**
35: $\mathcal{B}_{B.height+1} \leftarrow$ All blocks in D of height $B.height + 1$.
36: BROADCAST$(<$ UNRECEIVED$, B, \mathcal{B}_{B.height+1} >)$

end if
end for
return D
end procedure

procedure PROCESSCONFLICTMESSAGESSLOT(t, D)
  for all new $<$ CONFLICTINIT, B', $B>$ messages **do**
    if CONFLICTINIT not sent for $B'.author$ and $B'.height$ **then**
      BROADCAST($<$ CONFLICTINIT, B', $B>$
      BYZAGREEMENT(B')
    end if
  end for
  for all new $<$ UNRECEIVED, $B, \mathcal{B}_{B.height+1}>$ messages **do**
    $\mathbf{B}'_{B.height+1} \leftarrow$ All blocks of height $B.height + 1$
    if UNRECEIVED not sent for B **then**
      BROADCAST($<$ UNRECEIVED, $B, \mathcal{B}'_{B.height+1}>$)
    end if
    if B referenced by minority of blocks in $\mathbf{B}'_{B.height+1}$ **then**
      $D \leftarrow D \setminus B$
    end if
  end for
  upon BYZAGREEMENTOUTPUT() **do**
    $B_{out} \leftarrow$ GETBYZAGREEMENTVALUE()
    $\mathcal{B}_{del} \leftarrow$ {All blocks of height B_{out} proposed by $B_{out}.author$}
    $D \leftarrow D \setminus \mathcal{B}_{del}$
  return D
end procedure

The security of Algorithm 1 comes from the guarantee that conflicts will be detected for all honest nodes and handled with BYZAGREEMENT. Similarly, UNRECEIVED guarantee that honest nodes can discard or keep blocks that were not properly broadcast. The security analysis is given in Section 4.6.1.

THEOREM 8. ***P***$_{sd}$ *is a structured dissemination protocol under* $(\mathcal{A}_2(\beta_2), \mathcal{Z}_2)$ *where* $\beta_2 < 1/2$ *and* $T_b = 15\Delta$.

Proof. See Section 4.6.1.

We also show that Algorithm 1 can recover security after GST even if it was previously not secure due to an unstable network.

THEOREM 9. *$\boldsymbol{P}_{sd}$ is a structured dissemination protocol under $(\mathcal{A}_1(\beta_1), \mathcal{Z}_1)$ where $\beta_2 < 1/2$ after GST.*

Proof. See Section 4.6.1.

4.4 Graded common prefix

In this section, we utilize the block-DAG structure to introduce a novel primitive, the Graded Common Prefix. This primitive is weaker than standard consensus but still achieves finality for a subset of the blocks in a block-DAG, in partially synchronous networks. Unlike Graded Byzantine Agreement [**99**] that considers single values, graded common prefix considers block-DAGs, represented by sets of values, and outputs the common prefix amongst honest node's block-DAGs.

Intuition. To weaken standard consensus, agreement is relaxed by introducing grades 1 and 2. An output of grade 2 must be obtained with grade ≥ 1 for all honest nodes and no output of grade 2 can conflict. As outputs are block-DAGs, an output may be a subset of another, hence different but not conflicting.

By running GCP sequentially, a block-DAG D of grade 2 can be considered final even if some honest nodes have only obtained grade ≥ 1, as all honest nodes will propose D in the subsequent instance.

4.4.1 Definition

We define graded common prefix that is specific to block-DAG inputs and outputs. We separate the *weak validity* property from validity as it is achievable before $\max\{GST, GAT\}$ by a construction such as Algorithm 2 while validity is achieved after $\max\{GST, GAT\}$.

DEFINITION 19 (Graded Common Prefix. (GCP)). *In a graded common prefix protocol, every node $i \in 1, ..., n$ can input a block-DAG D_i, and outputs a graded block-DAG where each block a grade $\{1, 2\}$. A graded common prefix protocol satisfies the following properties:*

- *Termination: Every honest node terminates.*
- *Validity: If all honest nodes propose a block-DAG containing block-DAG D, $D' \supseteq D$ holds for any output D' by an honest node with grade 2.*
- *Weak Validity: If all honest nodes propose a block-DAG containing block-DAG D, $D' \supseteq D$ holds for any output D' by an honest node with grade ≥ 1.*
- *Agreement: If an honest node outputs grade 2 for a DAG D then all honest nodes output D and with grade ≥ 1, and no honest node outputs a DAG $D' \nsim D$ with grade 2.*

According the the properties of graded common prefix, different honest nodes may output different block-DAGs and different grades for the same blocks.

We additionally provide a generic version where nodes input a set of arbitrary values, such that the output is a graded set in Definition 20 that may be of independent interest for BFT.

DEFINITION 20 (Generic Graded Common Subset.). *A generic graded common prefix protocol takes as input a set $\mathcal{D}_i$ of values for every node $i \in 1, ..., n$, and outputs either a set of values with grade 1 or a set of values with grade 1 and a set of values with grade 2. A generic graded common prefix protocol satisfies the following properties:*

- *Termination: Every honest node terminates.*
- *Validity: If all honest nodes propose a set of values containing the value v, $v \in \mathcal{D}'$ holds for any output $\mathcal{D}'$ by an honest node with grade 2.*

- *Weak Validity: If all honest nodes propose a set of values containing the value v, $v \in \mathcal{D}'$ holds for any output $\mathcal{D}'$ by an honest node with grade ≥ 1.*
- *Agreement: If an honest node u outputs grade 2 for a set $\mathcal{D}$ of values then all honest nodes output $\mathcal{D}$ and with grade ≥ 1, and no honest node outputs a set $\mathcal{D}'$ with grade 2 where $(\mathcal{D}' \nsubseteq \mathcal{D}) \wedge (\mathcal{D} \nsubseteq \mathcal{D}')$.*

4.4.2 Partially synchronous graded common prefix

We propose a protocol to solve graded common prefix in a partially synchronous network for $n > 3f$ nodes, defined as pseudocode in algorithm 2.

Protocol flow. We provide a description of the protocol from Algorithm 2. First, a node u broadcasts its input D_u to all other nodes in a *propose* message (Line 2). Second, nodes wait for at least $n - f$ *propose* messages, and broadcast a *vote* with a common prefix block-DAG D_{vote} including every block-DAG contained in at least $n - f$ propose messages, or containing $\varnothing$ otherwise (Line 3-9). Figure 4.3 illustrates how nodes obtain a common prefix block-DAG. Thirdly, honest nodes wait for at least $n - f$ vote messages. An honest node outputs a block-DAG D_2 with grade 2 that includes every block-DAG contained in at least $n - f$ vote messages (Line 11-12). An honest node outputs a block-DAG D_1 with grade 1 that includes every block-DAG contained in at least 1 vote message (Line 13-14). Both D_1 and D_2 may be $\varnothing$.

Security intuition. We provide some intuition on the components of our protocol that guarantee GCP agreement. For D to be included in the output D_2 with grade 2, a quorum of $n - f$ vote messages containing D is required. With $n > 3f$ nodes, the quorum implies that all honest node receive at least 1 vote message containing D even when the network is asynchronous before GST. A single vote should therefore imply an output of grade 1 to guarantee agreement at all times.

The quorum requirement also ensures that a quorum for conflicting block-DAG is not obtainable, achieving the second agreement condition. However Byzantine nodes may attempt to broadcast votes for an arbitrary block-DAG D', that would also obtain grade 1. To prevent

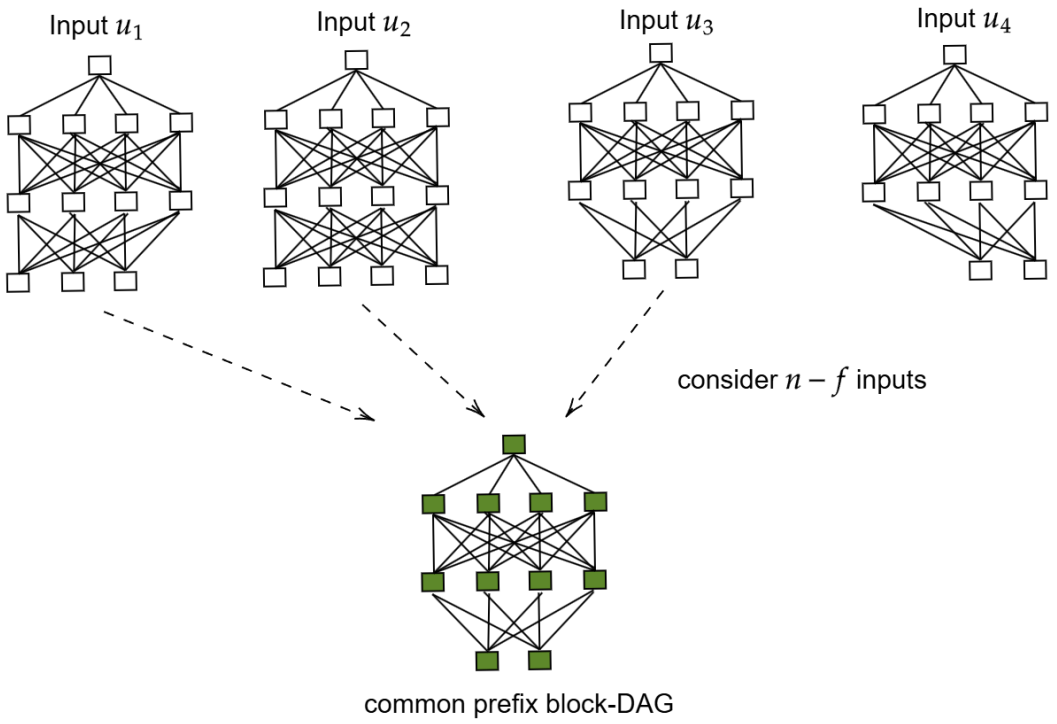


FIGURE 4.3. Example common prefix block-DAG obtained from $n - f$ received block-DAG inputs with $n = 4$ and $f = 1$. The blocks included in the common prefix are blocks included in at least $n - f$ of considered inputs.

this scenario, a valid vote for D contains the set of $n - f$ received propose messages whose block-DAGs include D.

THEOREM 10. *Algorithm 2 achieves the properties of graded common prefix as follows: validity after* $\max\{GST, GAT\}$*, termination in* 2 *network delays after* GAT*, and agreement and weak validity for* $n \geq 3f+1$ *nodes and up to* f *Byzantine nodes in a partially synchronous network.*

Proof. To show the security of Algorithm 2, we use quorum intersection arguments for each property of GCP. See Section 4.6.2 for the full proof.

Algorithm 2 Partially synchronous graded common prefix for a node u with input D_u.

1: **procedure** PSGRADEDCOMMONPREFIX(D_u)
2: BROADCAST($<$ PROPOSE, $D_u >$)

3: **upon** receiving $n - f$ PROPOSE messages **do**
4: **if** $\nexists D$ s.t. at least $n - f$ PROPOSE messages include D **then**
5: BROADCAST($<$ VOTE, $\varnothing >$)
6: **else**
7: $D_{vote} \leftarrow \bigcup\{D | D$ included in at least $n - f$ PROPOSE messages $\}$
8: $\Sigma_{vote} \leftarrow$ {All PROPOSE messages including D s.t. $D \subseteq D_{vote}\}$
9: BROADCAST($<$ VOTE, $D_{vote}, \Sigma_{vote} >$)
10: **end if**
11: **upon** receiving $n - f$ VOTE messages **do**
12: $D_2 \leftarrow \bigcup\{D | D$ included in at least $n - f$ VOTE message $\}$
13: OUTPUTGRADE2(D_2)
14: $D_1 \leftarrow \bigcup\{D | D$ included in at least 1 VOTE message $\}$
15: OUTPUTGRADE1(D_1)
16: **end procedure**

4.5 Flexible protocol with block-DAG

Simultaneously achieving dynamic availability and partition tolerance is a known impossibility due to the CAP theorem [**68**]. However, it is possible to enable clients to decide between prioritizing liveness or safety by running two sub-protocols concurrently. We adapt the Ebb-and-Flow framework [**104**] to obtain such a flexible protocols. Proposed constructions run a dynamically available protocol for liveness-prioritizing clients, and a partially synchronous BFT protocol for safety-prioritizing clients. In total, consensus is performed twice, once in each setting.

We present the class of secure Ebb-and-Flow protocols $\mathbf{P}$ that leverages a partially synchronous graded common prefix protocol $\mathbf{P}_{gcp}$ such as Algorithm 2, in parallel with a standard black box dynamically available protocol $\mathbf{P}_{da}$, of which many instantiations are available in the literature [**20**, **22**, **44**, **102**]. $\mathbf{P}_{da}$ is not limited to longest chain protocols, and can also be instantiated with potentially high throughput structured dissemination protocols presented in Section 4.3.1.

We therefore avoid performing standard consensus twice, as $\mathbf{P}_{gcp}$ leverages the information contained in the block-DAG (blockchains being special case of block-DAGs) created by $\mathbf{P}_{da}$.

Protocol instantiation. We assume the ledger LOG_{da} produced by $\mathbf{P}_{da}$ to be secure in the system model $(\mathcal{A}_2(\beta_2), \mathcal{Z}_2)$ and in the system model $(\mathcal{A}_1(\beta_1), \mathcal{Z}_1)$ only after GST. This is the case for our structured dissemination protocol in Section 4.3.1 as shown in Theorem 8 and Theorem 9. Alternatively, longest chain protocols have also been shown to be secure in both system models [**104**].

We also assume $\mathbf{P}_{gcp}$ to solve graded common prefix in the system model $(\mathcal{A}_1(\beta_1), \mathcal{Z}_1)$ and output a grade 1 block-DAG D_{g1} and a grade 2 output D_{fin}.

Protocol flow. Transactions are input to $\mathbf{P}_{da}$, which outputs LOG_{da} and a confirmed block-DAG D^t_{da}. If all calls to $\mathbf{P}_{gcp}$ have terminated, and $D^t_{da}.height > D_{fin}.height$ for the finalized block-DAG D_{fin}, nodes input the union of D^t_{da} and the last grade 1 output D_{g1} to the graded common prefix protocol $\mathbf{P}_{gcp}$. A grade 2 output D_{fin} of $\mathbf{P}_{gcp}$ becomes the finalized ledger LOG_{fin}, and a grade 1 output is saved in local state as D_{g1} (Line 13-16). LOG_{da} is obtained by the union of D_{fin} and D^t_{da} and *sanitizing* —removing any duplicate blocks as proposed in detail by Neu et al. [**104**] for the Ebb-and-Flow framework.

The protocol flow is illustrated in Figure 4.4.

Security Intuition.

The secure finality with $\mathbf{P}_{gcp}$ comes from its agreement and weak validity properties. By guaranteeing that all honest nodes output D with grade ≥ 1 if D is output with grade 2 due to agreement, all honest nodes will include D in their input for next instance. As a result, weak validity ensures that any future output will include D. The security of $\mathbf{P}_{da}$ is easier as Ebb-and-Flow protocols with a secure dynamically available protocol have already been proven [**104**]. The full proof is included in Section 4.6.3.

THEOREM 11. $\mathbf{P}$ *is a secure* $(\frac{1}{3}, \frac{1}{2})$*-Ebb-and-Flow protocol*

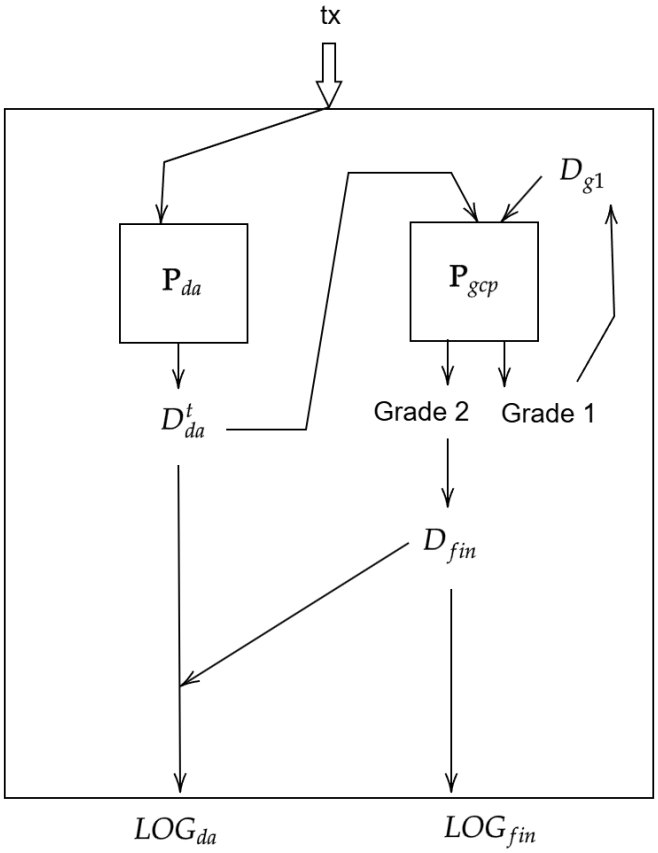


FIGURE 4.4. Sequence diagram for ebb-and-flow protocol $\mathbf{P}$ with a dynamically available protocol $\mathbf{P}_{da}$ and graded common prefix protocol $\mathbf{P}_{gcp}$.

Proof. We prove that $\mathbf{P}$ is a secure Ebb-and-Flow protocol, analyzing the security of each sub-protocol is shown under both system models. See Section 4.6.3 for the full proof.

Algorithm 3 $\mathbf{P}$ protocol using a dynamically available protocol $\mathbf{P}_{da}$ and graded common prefix $\mathbf{P}_{gcp}$

```
1: procedure GCPSLOT(LOG)
2:     if D_fin.height < D.height and ALLGCPTERMINATED() then
3:         PSGRADEDCOMMONPREFIX(D)
4:     end if
5: end procedure
6: procedure MAIN()
7:     if D_fin.height < D.height and ALLGCPTERMINATED() then
8:         PSGRADEDCOMMONPREFIX(D)
9:     end if
```

10: DASLOT(t)
11: $D^t_{da} \leftarrow$ DACONFIRMEDDAG(t)
12: GCPSLOT(D^t_{da})
13: $D_{g1} \leftarrow$ GCPGRADE1DAGS(t)
14: $D_{fin} \leftarrow$ GCPGRADE2DAGS(t)
15: **end for**
16: **end procedure**

4.5.1 Generalizing GAT with the MCAB model

In a real system, the assumption that GAT exists, the time after which all honest nodes are awake, may be considered too strong. This is notably the case in public blockchain deployments over the internet where nodes as public servers may always be targeted by denial-of-service attacks.

From the perspective of protocol design, the GAT assumption guarantees that enough honest nodes participate, for example to reach a quorum of votes. We highlight that *enough* awake honest nodes does not mean that *all* honest nodes must be awake.

To weaken the GAT assumption while still providing a time with *enough* participation, we define a system model using the mobile crashes of the MCAB model [**111**]. We obtain a more fine-grained treatment of GAT, the time from which participation is lower bounded, which we call Quorum Awake Time (QAT).

4.5.1.1 Definitions

Mobile crashes. Nodes that are mobile crashed are nodes whose outgoing and incoming messages are not delivered. Mobile crashed nodes may become cured, causing all previously undelivered outgoing and incoming messages to deliver.

Node corruptions. Up to $f + c$ nodes may be corrupted by the adversary, where f is the number of adaptive Byzantine faults, nodes gradually chosen by the adversary during protocol execution that may behave arbitrarily, and c is the number of mobile crashed nodes chosen

by the adversary for every slot. A node that was crashed in slot t and not in slot $t+1$ eventually receives all incoming messages from slot t. The remaining nodes are *honest*, and behave according to protocol specifications. For ease of notation w.l.o.g. we consider nodes numbered with $i \in [1, n-(f+c)]$ to be honest.

Quorum Awake Time. There is an unknown time after which $f + c \leq n \cdot \alpha$ called *Quorum Awake time* (QAT) for a resilience fraction α. QAT is decided by the adversary, and unknown to honest nodes.

Two system models. System model $(\mathcal{A}_1(\alpha)', \mathcal{Z}_1)$: In this system model, the network is partially synchronous, $c \leq n-1$ and $f \leq (n-c) \cdot \alpha$, and there exists a bounded QAT and a bounded GST.

System model $(\mathcal{A}_2(\alpha)', \mathcal{Z}_2)$: In this system model, the network is always synchronous (GST=0), $c \leq n-1$ and $f \leq (n-c) \cdot \alpha$, and QAT can be unbounded (QAT $\to \infty$).

Ebb-and-Flow with MCAB liveness We say that a (β_1, β_2)-secure Ebb-and-Flow protocol has *MCAB liveness* if finality holds under $(\mathcal{A}_1(\alpha)', \mathcal{Z}_1)$ and $(\mathcal{A}_1(\alpha), \mathcal{Z}_1)$, and dynamic availability holds under $(\mathcal{A}_2(\alpha)', \mathcal{Z}_2)$ and $(\mathcal{A}_2(\alpha), \mathcal{Z}_2)$.

4.5.1.2 Comparison with Ebb-and-Flow

Mobile crashes behave in the same way as adaptive sleepiness from the sleepy model [**107**]: both allow the adversary to prevent a node's messages from delivering, and allow a node to be cured/awoken to become honest again. In the sleepy model, there is no parameter for the number of simultaneous sleepy nodes, while the MCAB model [**111**] provides a fine-grained number of mobile crashes at any time.

As a result, the key difference with the system models of Ebb-and-Flow [**104**]'s is that after QAT in our models, there is still the possibility of mobile crashes, unlike after global awake time where the all nodes are either honest or Byzantine. Note that $(\mathcal{A}_2(\alpha)', \mathcal{Z}_2)$ describes the same model as $(\mathcal{A}_2(\alpha), \mathcal{Z}_2)$ if QAT $\to \infty$, and that setting $c = 0$ after QAT

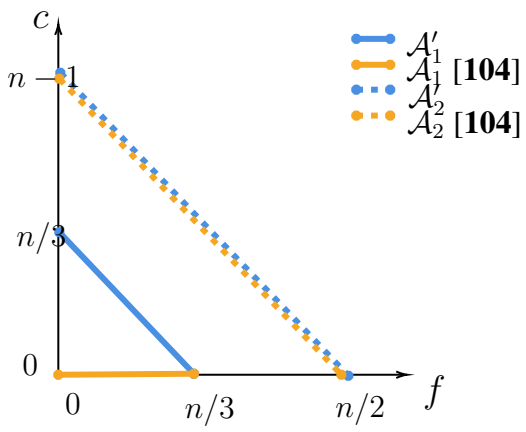


FIGURE 4.5. The highest number c of mobile crash faults and the number f of Byzantine faults allowed for protocols against $\mathcal{A}'_1$ after QAT and $\mathcal{A}'_2$ in our work, and their respective equivalents from Ebb-and-Flow $\mathcal{A}_1$ after GAT and $\mathcal{A}_2$ [**104**]

.

for $(\mathcal{A}'_1(\alpha), \mathcal{Z}_1)$ results in $(\mathcal{A}_1(\alpha), \mathcal{Z}_1)$ We visualize the difference between $(\mathcal{A}_2(\alpha)', \mathcal{Z}_2)$, $(\mathcal{A}_2(\alpha), \mathcal{Z}_2)$, $(\mathcal{A}_1(\alpha), \mathcal{Z}_1)$, and $(\mathcal{A}_1(\alpha)', \mathcal{Z}_1)$ in Figure 4.5.

4.5.1.3 Our protocols in the MCAB model

Deterministic leader based protocols such as Hotstuff have been shown to lose liveness in the presence of mobile crashes [**111**], implying that even after $\max(QAT, GST)$, Snap-and-Chat protocols using Hotstuff or Streamlet are not able to guarantee liveness. In contrast, every step in our constructions for GCP involve every node performing the same protocol step. It is easy to see that these fit the criteria for *abundant roles*, shown to be sufficient for liveness in the MCAB model [**111**]. Our protocol $\mathbf{P}$ is therefore an Ebb-and-Flow protocol with MCAB liveness, extending our results in Theorem 8 to $\mathcal{A}'_2$.

Similarly, Theorem 11 can be extended to $\mathcal{A}'_1$ and $\mathcal{A}'_2$ as long as the sub-protocols $\mathbf{P}_{da}$ and $\mathbf{P}_{fin}$ only include protocol steps that are *abundant roles* or *concealed roles*, which guarantees liveness in the MCAB model [**111**].

4.6 Security analysis

4.6.1 Constant latency dynamically available structured dissemination

To analyze the security of $\mathbf{P}_{sd}$, we analyze how conflicts and late blocks are resolved under $(\mathcal{A}_2(\beta_2), \mathcal{Z}_2)$ where $\beta_2 < 1/2$ before showing that our full protocol achieves structured dissemination.

LEMMA 4. *Unreceived message agreement. If an honest node discards a block B of height h after receiving an* UNRECEIVED *message, all honest nodes discard B.*

Proof. We proceed by contradiction. Assume u discards B and, w.l.o.g., is the first honest node that broadcasts an UNRECEIVED message at time $t_u \geq t + 2\Delta$ where t is the time B was proposed. Also assume that v does not discard B until it is consistent. Since v does not discard B, we have two cases:

- Case 1: v has not received any UNRECEIVED message, a contradiction as u rebroadcasts UNRECEIVED message and it is received within Δ slots.
- Case 2: v has received at least $f_{t+\Delta} + 1$ blocks of height $h + 1$ referencing B by time $t + 2\Delta$, where t is the time honest nodes propose blocks of height h. This implies that at least one honest node has proposed a block of height $h + 1$ at time $t + \Delta$, received by all honest nodes before $t + 2\Delta$ by the synchronous assumption. As a result, u receives B before $t + 2\Delta < t_u$, and does not send a UNRECEIVED message, a contradiction.

LEMMA 5. *Timing of conflict instance. For any two honest nodes starting* BYZAGREEMENT *for an equivocating node $B.author$ at times t, t', we have $|t - t'| \leq \Delta$.*

Proof. Honest nodes always broadcast a CONFLICTINIT message before starting BYZAGREEMENT for an equivocating node $B.author$. By assumption, all honest nodes receive the CONFLICTINIT message in at most Δ slots. All honest nodes will therefore start BYZAGREEMENT at most Δ slots after the first honest node that starts BYZAGREEMENT.

LEMMA 6. *Security of* BYZAGREEMENT *Any instance of* BYZAGREEMENT *is secure in Algorithm 1 under* $(\mathcal{A}_2(\beta_2), \mathcal{Z}_2)$ *where* $\beta_2 < 1/2$ *and* $T_b = 15\Delta$.

Proof. Our conflict resolution protocol BYZAGREEMENT can be instantiated with a single view of the atomic broadcast protocol [**91**] in sleepy model with backward time $T_b = 11\Delta$. The confirmation of blocks of the same height h depends on the proposals at height h sent synchronously at time t, on the messages sent between t and $t + 3\Delta$, and all messages of BYZAGREEMENT instances where $B.height = h$. All honest nodes start instances of BYZAGREEMENT where $B.height = h$ between t and $t + 4\Delta$ by Lemma 5. We therefore have $T_b = 4\Delta + T_{bm} = 15\Delta$.

4.6.1.1 Full protocol

LEMMA 7. *Consistency.* $\boldsymbol{P}_{sd}$ *achieves* $\delta-$*consistency of a structured dissemination protocol under* $(\mathcal{A}_2(\beta_2), \mathcal{Z}_2)$ *where* $\beta_2 < 1/2$, $\delta = 3\Delta$, *and* $T_b = 15\Delta$.

Proof. Part 1: We proceed by contradiction in two parts. Assume a block B proposed by an arbitrary node $B.author$ of height h is consistent in an honest node u's block-DAG, but in another honest node v's block-DAG, no block by $B.author$ of height h is consistent. Since B is consistent for u, u has waited 2Δ time slots after proposing a block B_u of height $h+1$ such that $B_u \to B$ at time slot t_{h+1}. Since v is honest, v has proposed a block B_v at height $h+1$ at time t_{h+1}. If B was received before t_{h+1} by v, B_v references B, and considers B δ-consistent after time $t_{h+1} + 2\Delta$, a contradiction. If v has not received B before t_{h+1}, it is received before $t_{h+1} + \Delta$ from u through the synchronous network. v therefore broadcasts an UNRECEIVED message, received by all honest nodes before $t_{h+1} + 2\Delta$. By Lemma 4, if an honest node discards B due to the UNRECEIVED message, all nodes discard B a contradiction. If no honest node discards B, v will confirm B at time $t_{h+1} + 2\Delta$, a contradiction. This completes the first part of our proof.

Part 2: Assume a block B proposed by an arbitrary node $B.author$ of height h is consistent in an honest node u's block-DAG, but in another honest node v's block-DAG, a block $B' \neq B$ by $B.author$ of height h is consistent. Consider two cases for time t when u extends B with

a proposal and time t' when v receives B'. We assume w.l.o.g. that u is the first honest node receiving B and v is the first honest node receiving B':

- Case 1: $t' > t + 2\Delta$. All honest nodes received B as a parent of u's block of height $h + 1$ at time $t + \Delta$. As a result, when CONFLICTINIT message is broadcast after receiving B', all honest nodes propose B in BYZAGREEMENT and, by validity, B is output for all honest nodes and B' is discarded. BYZAGREEMENT is secure by lemma 6.
- Case 2: $t \leq t' \leq t + 2\Delta$. If all honest nodes receive B first, this is the same case as case 1. If some nodes may receive B first while others receive B' first, we will show that all honest nodes enter BYZAGREEMENT and that no two honest nodes will exclusively discard B, B' respectively. By time $t' + \Delta$, all honest nodes have received B from u and B' from v by the synchronous network assumption. Since $t' + \Delta \leq t + 2\Delta$, u receives B' before the confirmation timer for B runs out, and enters BYZAGREEMENT. By the agreement property of BYZAGREEMENT, all honest nodes obtain the same output B or B'.

Part 3: Assume a block B of height h is consistent at time t in an honest node u's block-DAG, but another honest node v confirms a block $B' \neq B$ of height h after t. Since B is confirmed for u, all instances of BYZAGREEMENT for blocks in height h have terminated. When receiving B', v broadcasts an UNRECEIVED message. However, as we have shown in Part 1 of the proof, if no honest node has confirmed B' by time t, B' will be discarded. Otherwise, B' is confirmed by all honest nodes by time t, a contradiction.

THEOREM. ***P***$_{sd}$ *is a structured dissemination protocol under* $(\mathcal{A}_2(\beta_2), \mathcal{Z}_2)$ *where* $\beta_2 < 1/2$ *and* $T_b = 15\Delta$.

Proof. δ-Consistency: Follows from Lemma 7.

τ, t-DAG growth: An honest node's block timer $BlockTimer()$ triggers every Δ_b time, upon which an honest node broadcasts a new block. Within t time, there at least $\frac{t}{\Delta_b} \cdot (n^*)$ additional blocks proposed by the n^* nodes that are honest and awake during t. In cases

with conflicts, the termination of BYZAGREEMENT guarantees that blocks of any height with honest proposals are eventually confirmed or discarded. We therefore have τ, t-DAG growth where $\tau = \frac{n^* - f}{\Delta_b}$.

μ-DAG quality: We first show by contradiction that at any height k for an honest node, there are up to n blocks proposed by distinct nodes. Assume that at height k, an honest node u includes $n' > n$ blocks. This either implies that there are n' distinct nodes, a contradiction, or that u has included more than one block of height k proposed by the same node. This is also a contradiction as the protocol is consistent.

External Integrity: External integrity is guaranteed as honest nodes only add a block B to their local block-DAG if $condition(B) = 1$ holds.

4.6.1.2 Self-healing DAG after GST

We provide the proofs for $\mathbf{P}_{sd}$'s security after GST under $(\mathcal{A}_1(\beta_1), \mathcal{Z}_1)$.

LEMMA 8. *Consistency of blocks before GST after GST. All blocks proposed before GST under $(\mathcal{A}_1(\beta_1), \mathcal{Z}_1)$ where $\beta_1 < 1/2$ are eventually discarded or consistent for all honest nodes after GST.*

Proof. At time $t > GST$, consider the set $\mathcal{C}$ containing every pair of blocks B and $B' \neq B$ of the same height where $B.author = B'.author$, and B is confirmed for an honest node u while B' is confirmed for another honest node v. We also consider the set $\mathcal{M}$ containing every block B of height h such that B is confirmed in an honest node u's block-DAG, but in another honest node v's block-DAG, no block by $B.author$ of height h is confirmed.

For pairs in $\mathcal{C}$, consider the following proof by contradiction. Assume a block B proposed by an arbitrary node $B.author$ of height h is consistent in an honest node u's block-DAG, but in another honest node v's block-DAG, a block $B' \neq B$ by $B.author$ of height h is consistent. Since the network is synchronous after GST, Lemma 5 holds and all honest nodes enter an instance of BYZAGREEMENT. Lemma 6 also holds since BYZAGREEMENT is started after

GST and the network is synchronous. By the agreement property of BYZAGREEMENT, all honest nodes obtain the same output B or B', a contradiction.

Honest nodes therefore eventually agree on a consistent block for all pairs in $\mathcal{C}$.

For blocks in $\mathcal{M}$, consider the following recursive proof once $\mathcal{C} = \varnothing$. We order the blocks in $\mathcal{M}$ by height such that the block with the largest height has index $i = 0$. Blocks of the same height have neighboring indexes.

For $i = 0$, we denote the block B_0 of height h_0. There are no blocks of larger height in $\mathcal{M}$ by definition, and blocks of height $h_0 + 1$ are confirmed since $\mathcal{C} = \varnothing$. Once an honest node broadcasts an UNRECEIVED message after receiving B_0 after GST at time t_u, all honest nodes receive the UNRECEIVED message after $t + \Delta$ and either discard or keep B_0 after $t + 3\Delta$. We prove by contradiction that all honest nodes discard B_0 if an honest node u discads B_0. Since u discards B_0, it has received blocks of height $h_0 + 1$ such that a majority does not reference B_0. Since $\mathcal{C} = \varnothing$ and B_0 has the highest height in $\mathcal{M}$, the blocks of height $h_0 + 1$ are confirmed for all honest nodes, a contradiction as they would all discard B_0 too. As a result, all honest nodes either discard or confirm B_0 after GST. For the recursion proof, we now replace B_0 with the empty element $\varnothing$ in $\mathcal{M}$, maintaining the same indexes for the other elements.

For $i + 1$, blocks of height h_i in $\mathcal{M}$ are either discarded or confirmed. The block B_{i+1} is therefore the block with the largest height in $\mathcal{M}$. Using the same proof as for $i = 0$, all honest nodes either discard or confirm B_{i+1} after GST, completing the recursive proof.

THEOREM. ***P***$_{sd}$ *satisfies a structured dissemination protocol's properties under* $(\mathcal{A}_1(\beta_1), \mathcal{Z}_1)$ *where* $\beta_1 < 1/2$ *after GST.*

Proof. Before GST, δ-Consistency, τ, t-DAG growth, and μ-DAG quality are not guaranteed and honest nodes may have differing views of confirmed block-DAGs, with no guaranteed growth rate or fraction of honest blocks. External integrity is still guaranteed as honest nodes only add a block B to their local block-DAG if $condition(B) = 1$ holds, irrespective of network conditions.

We now analyze each property after GST. Compared to the proofs for Theorem 8, the system model is equivalent as both networks are synchronous, and the fraction of Byzantine nodes is bounded by $1/2$. The only difference with the proofs for Theorem 8 is therefore that honest nodes don't start with the same consistent block-DAG. We now show that δ-Consistency, τ, t-DAG growth, and μ-DAG quality eventually hold after GST.

τ, t-DAG growth: The proof is the same as for Theorem 8.

δ-Consistency: Since before GST, honest nodes have differing confirmed block-DAGs, and are eventually discarded or confirmed as shown by Lemma 8. For blocks proposed after GST, the proof is the same as for Lemma 7.

μ-DAG quality: The proof is the same as for Theorem 8, since we have shown that the protocol is consistent after GST.

4.6.2 Graded Common Prefix

THEOREM. *Algorithm 2 achieves the properties of graded common prefix as follows: validity after* $\max\{GST, GAT\}$*, termination in* 2 *network delays after* GAT*, and agreement and weak validity for* $n \geq 3f + 1$ *nodes and up to* f *Byzantine nodes in a partially synchronous network.*

Proof. Validity. For a block-DAG D, that is a subset of all honest proposals, it is included in $n - f$ $Proposal$ messages after GAT and which are received by all honest nodes after waiting Δ time slots after GST. All nodes therefore include D in their VOTE messages, and all honest nodes therefore receive $n - f$ vote messages containing D after Δ time slots, hence including D in their grade 2 output.

Weak Validity: We proceed by contradiction. Assume that a block-DAG D is a subset of all honest proposals and an honest node u outputs D' with grade ≥ 1 such that $D \nsubseteq D'$. To output D' with grade ≥ 1, u has received at least one VOTE message where $n - f$ proposals

don't include D. This is a contradiction as all honest nodes include D in their proposal, and only up to f nodes are Byzantine.

Agreement. We proceed by contradiction. Assume that an honest node u has included a block-DAG D in its grade 2 output, and that another honest node doesn't include D in its output ≥ 1 at protocol termination. For u to include D in its grade 2 output, at least $f+1$ honest nodes have sent a VOTE messages whose block-DAG $D_i \supseteq D$ $i \in \{1, ...f+1\}$, of which at least 1 is included by u in its set of $2f+1$ vote messages. This is a contradiction as the honest node v would include D with grade at least 1.

Assume that an honest node u has decided D with grade 2, that another honest node v decides $D' \nsim D$ with grade 2. To decide D with grade 2 and not D', u has received $n-f$ votes that contain D and not D'. By quorum intersection, v can not obtain $n-f$ votes that also contain D' unless $D' \sim D$, a contradiction.

Termination: Termination is guaranteed in 2 network delays, or $2 \cdot \Delta$ after GST, as honest nodes receive at least $n-f$ proposals and $n-f$ votes since $n-f$ nodes are honest and awake.

4.6.3 Ebb-and-Flow using Graded Common Prefix

We prove that $\mathbf{P}$ is a secure Ebb-and-Flow protocol, using a similar proof structure as for snap-and-chat protocol [**104**].

THEOREM. $\mathbf{P}$ *is a secure* $(\frac{1}{3}, \frac{1}{2})$*-Ebb-and-Flow protocol*

Proof. We first show the safety of LOG_{fin} under $(\mathcal{A}_1(\beta_1), \mathcal{Z}_1)$ where $\beta_1 = \frac{1}{3}$. Due to agreement of graded common prefix, when an honest node outputs a final block-DAG B with grade 2, all honest nodes output B with grade at least 1. As a result, every honest node i will input $B_i \supset B$ to the next instance of graded common prefix. By validity, $B \subseteq D$ for a block-DAG output D of $B \nsubseteq D$ with grade 2.

For the liveness of LOG_{fin} after time $\max\{GST, GAT\}$ under $(\mathcal{A}_1(\beta_1), \mathcal{Z}_1)$, consider the following: After time $\max\{GST, GAT\}$, $\mathbf{P}_{da}$ is safe and live and all nodes input the same latest confirmed block-DAG B from $\mathbf{P}_{da}$. The validity property of $\mathbf{P}_{gcp}$ then ensures that B is included in the output with grade 2.

To show the security of $\mathbf{P}$ under $(\mathcal{A}_2(\beta_2), \mathcal{Z}_2)$ where $\beta_2 = \frac{1}{2}$, $\mathbf{P}_{da}$ must be secure and its output LOG_{da} must be consistent with the finalized output LOG_{fin}. The security of $\mathbf{P}_{da}$ follows from our black box secure dynamically available protocol assumption.

An output D_{fin} from $\mathbf{P}_{gcp}$ requires at least one honest input that includes D_{fin} to be output with grade 2. Since $\mathbf{P}_{da}$ is secure, D_{fin} is included in the confirmed block-DAG output D_{da} and $D_{fin} \subseteq D_{da}$. LOG_{fin} is extracted from D_{fin}, and LOG_{da} from $D_{da} \cup D_{fin} = D_{da}$. LOG_{fin} is therefore consistent with LOG_{da}.

4.7 Experimental evaluation

We evaluate our protocols experimentally to show the following:

(1) Our DAG protocol $\mathbf{P}_{sd}$ for the sleepy model throughput scales with the number of nodes, while current state-of-the-art sleepy model BFT does not. In addition, our protocol's common case latency is lower.
(2) The latency of our graded common prefix protocol is not affected by crashed nodes, unlike existing leader based BFT such as Hotstuff [**130**].

4.7.1 Setup

We implemented both our protocols and their respective baselines in Haskell using the *distributed-haskell* library[1] to instantiate asynchronous nodes and their communication network. The code is available online[2]. We ran our experiments on a national research cloud,

[1]https://github.com/haskell-distributed
[2]https://github.com/hans-repo/Consensus

using 10 m3.xsmall instances which have with 2GB of RAM and one vCPU. Each instance ran up to 6 nodes for a total of 60 nodes in our experiments.

In each protocols, arbitrary data is proposed by nodes, and we consider 8B of data as 1 operation. For each committed block of operations, its latency and the current throughput is tracked. Throughput measures the number of committed operations divided by the elapsed time of the experiment. Latency measures the elapsed time between the moment a operation is proposed by a node, until the moment a client receives its confirmation. As different nodes may commit a operation at slightly different times, the average is computed.

4.7.2 Dynamically available protocols

Since constant-latency BFT in the sleepy model has no publicly available implementation to our knowledge, we implemented a prototype for sleepy model BFT based on Mahkhi et al.'s sleepy BFT [**91**] as our baseline.

To show that our protocol scales with participating nodes, we increase the number of nodes from 10 to 60 in steps of 10 (1 additional node per instance), and run each protocol for 600 seconds. Blocks are set to contain 5 operations, for a total of $200B$ per block. The protocol's timers (Δ) were set to 1 second, such that nodes collect messages for 1 second before proceeding to the next step.

Results. As depicted in Figure 4.6a, the baseline protocol's throughput remains relatively stable and obtains no increase with more participating nodes. The performance of the baseline protocol does not noticeably degrade either due to the relatively large timeout set to 1 second, meaning that messages have more than enough time to deliver between protocol steps.

In contrast, in our dynamic DAG protocol, new blocks are proposed every $\Delta = 1s$ which inherently contributes a x4 increase in throughput, compared to the baseline that proposes a block every 4Δ. There is also a block for each node proposed, and we observe an initial increase in throughput as the number of nodes increases. Note that a pipelined version of the baseline could potentially mitigate the x4 factor difference, proposing every Δ instead.

At 60 nodes, our instances' modest resources (CPU, RAM, Network) were saturated and throughput drops. We expect stronger machines to be able to support more throughput before saturation, as is the case in DAG BFT works with access to such resources [**46**, **121**].

As shown in Figure 4.6b, our protocol exhibits slightly lower latency, almost 1 second less than the baseline. With timers set to $\Delta = 1$ second, these results align with the theoretical expectations outlined in Table 4.1, where our protocol commits in 3Δ, compared to the baseline's 4Δ. For reference, in simulations for DAGs with participation-dependent latency, 22 seconds was the best case in Prism [**22**].

Variance in our latency measures can be explained by temporary clock de-synchronization between our instances as the protocols run, causing occasional drift in the latency measurement. However, the highest statistical standard deviation of our latencies was ~ 0.1s, and these inacurracies do not prevent us from observing the expected 1 second lower latency.

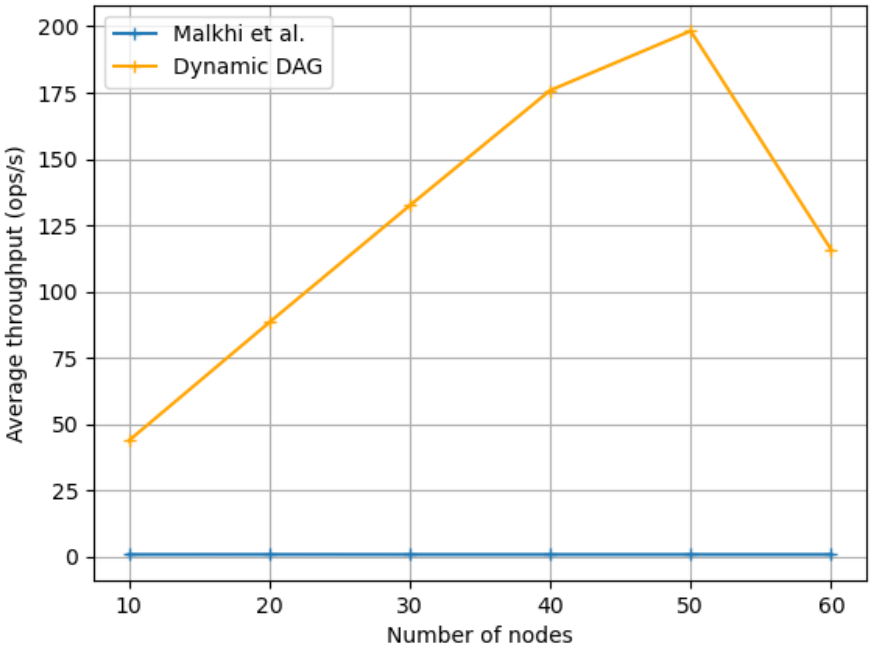
4.7.3 Partially synchronous protocols

To fairly compare our GCP protocol to a partially synchronous BFT, we implemented Hotstuff [**130**] under the same Haskell framework and *distributed-haskell* library.

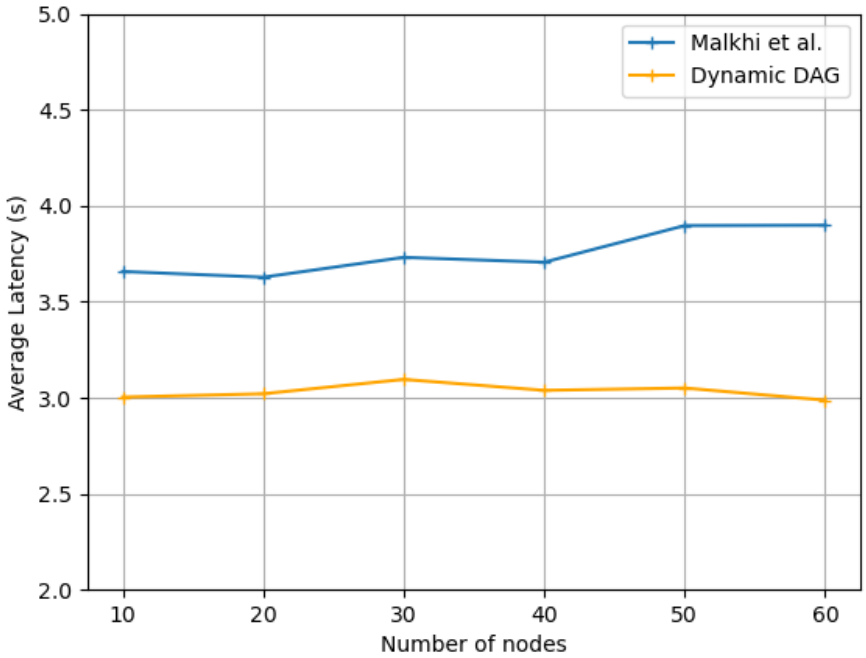
To show the impact of crashes on latency for our GCP protocol and a leader based BFT like Hotstuff, we increase the number of crashes from 0 to 6 in steps of 1, and run each protocol for 300 seconds. Each of the 10 instances run 2 nodes for a total of 20 nodes. Hotstuff blocks are set to contain 10 operations, while GCP proposals contain 10 block-DAG identifiers, represented by 10 operations of the same size, for a total of 200B per block or proposal for each protocol. Hotstuff's timeout was set to 1 second.

Results. As depicted in Figure 4.7, in the baseline protocol that relies on leaders, crashes increasingly affect the average latency due to the expensive leader change operation, and waiting for timeout timers.

In contrast, the latency of our GCP is faster without crashes and remains stable as the crashes increase, matching our theoretical expected benefits of GCP.



(A) Throughput vs. number of nodes.



(B) Latency vs. number of nodes.

FIGURE 4.6. Throughput and latency evaluation with a gradually increasing number of nodes for our proposed DAG protocol and Malkhi et al.'s constant latency sleepy BFT [**91**].

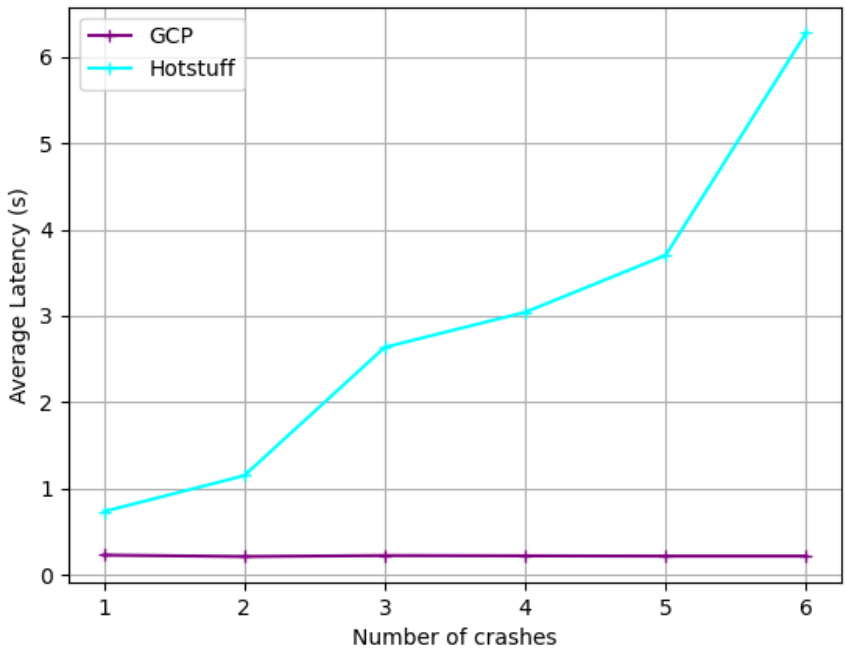


FIGURE 4.7. Latency evaluation with a gradually increasing number of crashes for our proposed GCP protocol and Hotstuff [**130**].

4.8 Conclusion

This work introduces the first constant latency dynamically available DAG protocol, and a new primitive, graded common prefix, to finalize block-DAGs in the partially synchronous setting without resorting to standard consensus. These findings are synthesized in a hybrid protocol consisting of the dynamically available DAG protocol, coupled with GCP. The hybrid protocol lets clients choose prioritizing liveness or safety. By leveraging the DAG structure in each sub-protocol, we obtain high throughput and finalization without the redundant work of running standard consensus again for asynchronous safety.

CHAPTER 5

Incentivizing fault independence.

5.1 Introduction

Byzantine Fault-Tolerant (BFT) protocols have been adopted in multiple public blockchains. The resilience of such protocols is usually given in terms of the number f of Byzantine faults it can tolerate. In the classic BFT setting, it is implied that these are f independent faults. How this independence is achieved is abstracted away and considered feasible in most classical settings involving a closed system of a few nodes.

The fault independence assumption is, however, less realistic in open blockchain networks. Nodes in a blockchain network participate by running a form of server with different *configuration* aspects, such as the hardware, software implementation, networking tools, etc. A single fault in a configuration aspect may simultaneously affect multiple nodes, or even all nodes. For example, there have already been cases where an entire blockchain halts due to a bug in the software that every [**57**, **87**] or most [**76**] nodes run.

Nevertheless, the importance of fault independence has not been ignored in Ethereum, the second largest blockchain project by market cap after Bitcoin as of today. The Ethereum consensus node and execution layer client software both have multiple implementations by different developer teams, motivated by the need for what is known as *client diversity*.

At the time of writing, according to self reported data [**4**], the top implementations (Geth and Lighthouse) for the consensus node and execution layer client make up over 41% and 42% of validators respectively. A single fault in either implementation is enough to stall new finalizations, as they require over $2/3$ of validators to be online to vote. This indicates that

simply providing multiple implementations does not meaningfully improve the resilience, one software fault can still affect more than f nodes.

Intuitively, to avoid the problem described above, nodes should have use a wider diversity of software implementations, minimizing the number of nodes that a single fault impacts. However, we first need to understand how to measure said diversity, and what the requirements are to improve resilience over the baseline, where a single fault can stall the system.

Once the diversity requirements are well defined, we also wish to understand how achieve them. Currently, Ethereum nodes still concentrate around the most popular implementations. While members of the Ethereum community push for node diversity, for example providing a "client switcher" [**5**], or through promotion by its founder [**32**], there are no tangible incentives for validators to use different implementations than the most popular and convenient ones.

The client diversity problem in Ethereum motivates us to explore mechanisms to increase diversity not just for client software, but any relevant configuration aspect. For our proposed mechanisms to be practical, we prefer mechanisms that do not require modifications to existing blockchain components such as the consensus protocol, and therefore operate as an additional layer of the system.

As such, we set the following goals:

(1) Clearly define the configuration diversity requirements for a blockchain to tolerate more than one configuration fault.
(2) Design and evaluate mechanisms to achieve the defined requirements, without modifying existing components of the blockchain.

5.1.1 Contributions.

Defining configuration diversity that improves resilience.

For goal 1), we define configurations in a distributed system as different *versions* (e.g. Geth implementations) of a *dimension* (e.g. client execution client).

In the baseline, a fault in a single version of a dimension can immediately affect more than f nodes. We call such a dimension a *single point of failure*. We show that over $\lceil n/f \rceil$ evenly distributed versions of a dimension are sufficient to avoid being a single point of failure. For example, at least 4 execution clients evenly distributed among nodes on Ethereum where $\lceil n/f \rceil = 4$, prevents the execution client dimension from being a single point of failure.

Focus on incentive mechanisms.

To address goal 2), a naive solution is to collect information from each node about a dimension's version, and assign an appropriate subset (at least $\lceil n/f \rceil$ and evenly distributed) to run consensus. The veracity of a node's stated version is proven by a remote attestation proof, for which different constructions exist for different server dimensions [**42**, **58**, **115**, **117**].

The overhead of such an approach is considerable, as every node needs to constantly provide a remote attestation. In addition, some nodes may never be selected to participate in consensus, arguably hurting open nature of a blockchain such as Ethereum.

We instead focus on methods that monetarily reward nodes for increasing configuration diversity. Incentive mechanisms can be deployed on an existing system, in parallel to other layers like consensus, or even any existing node reward mechanisms. Unlike directly assigning nodes with the right configuration to consensus, incentive mechanisms are opt-in: nodes are only incentivized to provide attestations and select versions that increase diversity. In addition, attestations can be provided less frequently than for every consensus decision.

Rewarding diversity and the unknown cost problem.

For nodes to spread out among different versions, they need an incentive to move away from the most popular version. In the case of Ethereum execution clients, the most popular version is likely popular because it is convenient, with easily available documentation and support resources.

To compensate nodes for moving to an execution client that is less convenient, we can allocate additional rewards to nodes running less popular execution clients. Informally, a rational node would compare the net gain $R(v) - C(v)$ for every available version v obtained with

the allocated reward $R(v)$ and the version specific costs $C(v)$. The cost $C(v)$ encapsulates the overhead involved with running a version v, including the documentation example.

Intuitively, increasing rewards $R(v)$ to compensate higher costs $C(v)$ makes sense. We can simulate the system with rational nodes, testing which reward assignments lead to the best diversity results.

The difficulty lies in the inability to quantify the multitude of factors involved in the overall cost $C(v)$ of a version. For example, different clients in Ethereum might be differently optimized, leading to higher energy costs to run. There may also be intangible costs, such as a smaller technical support community and incomplete documentation for certain client versions. The problem is that for a real system, we have no concrete information about real costs for different versions, even for costs that can be quantified more easily in principle like energy cost.

Instead of attempting to quantify the factors involved in the multitude of operational aspects of a distributed system's node, we suggest adaptively adjusting rewards for different versions. By observing the behavior of the nodes, indirect information about each version's cost, quantifiable or not, can be inferred.

Cost discovery using adaptive rewards.

To avoid dependency on unknown real costs, we propose systems that adjust reward pools for each configuration separately, based on their current popularity. Less popular configurations' rewards are boosted and more popular configurations' rewards are decreased. As configurations are initially more or less popular based on their real unknown cost, from quantifiable electricity bills to unquantifiable practicality elements like documentation, the rewards adjustments could be considered a kind of cost discovery tool for each configuration, gradually assigning more rewards to costlier versions.

To summarize, we want to reach a target optimal node distribution among different versions, we have a suboptimal starting node distribution, we can control reward allocation to each version, and we have little concrete information about the costs of each version.

As a result, we believe that we can leverage well-known control theory principles for systems where an analytical solution is unknown or untractable. More specifically, the Proportional Integral Derivative (PID) Controller is designed to to converge a state variable (current node distribution) to a target state (ideal node distribution) by adjusting an input variable (reward allocation).

Since we have little knowledge of the real system dynamics, we also believe that Reinforcement-Learning (RL) methods are well suited to allocate rewards for diversity. As RL methods are designed to adapt to more complex system dynamics than PID controllers, we expect them to perform better when more versions are involved. The RL methods are also considered to explore their benefits in a changing environments.

Evaluating the PID controller and RL methods.

In total, we design, compare, and evaluate 3 fault independence incentive mechanisms. The first is a traditional PID controller, tuned with the standard Ziegler-Nichols method. The second is a PID controller tuned with a RL agent, able to adapt the 3 PID parameters while the system runs. The third is a RL agent that directly sets rewards for each configuration, therefore handling a set of variable growing with the number of considered configurations.

We compare our 3 methods in a simulated environment, modeling the behavior of each node in the system w.r.t the system's allocated rewards and configuration costs. Our incentive mechanisms do not have access to the costs of each configuration, mirroring the lack of knowledge for real world configurations such as Ethereum execution client costs.

Our results show that the advantages of the RL approaches are only noticeable in extreme environment changes, and that the Ziegler-Nichols-tuned PID controller is able to otherwise perform slightly better than the other approaches. In addition, its implementation is simpler and the mechanism more tractable.

In summary, our contributions are the following:

- We define single points of failures in a distributed system and show the first formal sufficient conditions for a dimension of a distributed consensus system to avoid

being a single point of failure. A dimension must have at least 4 distinct versions, evenly distributed among nodes.

- We design 3 incentive mechanisms designed to avoid single points of failure.
- We extensively evaluate the 3 proposed incentive mechanisms and compare their tradeoffs in simulations with rational reward-maximizing nodes. Contrary to out initial beliefs, our simulations indicate that a traditional PID controller is better suited than the RL methods in most cases.

5.2 Definitions and theoretical analysis.

5.2.1 Definitions.

Nodes. Nodes represent participants, replicas, or servers taking part in a distributed system. We consider n nodes throughout this work.

Fault threshold. A protocol's fault threshold f is the number of nodes that may be faulty. If the number of faulty nodes is higher than f, the protocol's security is compromised.

Ideal attestation primitive. An ideal attestation primitive can output a proof $\pi_{v,i}$ for a version v and a node i if and only if i is using v.

Dimensions and versions. Each node is configured with $|\mathcal{D}|$ *dimensions*, and a dimension d is associated with the set $\mathcal{V}$ of *versions*. We use the terms version and *state* interchangeably. Dimensions represent an aspect of the node u's real life configuration, such as the execution client, while a version is the specific instance run by u, such as the geth client on Ethereum.

Impact. The *impact* p of a version v is the number of nodes configured with v. The sum of all versions' impact for a dimension is 1.

Diversity Measurement. We use the Shannon Entropy to measure diversity as introduced by Yu [**131**], generically defined as $H((p)) = -\sum_{i=1}^{n} p_i \log_2 p_i$ for a probability distribution **p**.

For diversity in our system model with the set $\mathcal{V}$ of versions $v_i \in \mathcal{V}$ with impact p_i, we obtain $H = -\sum_{i=1}^{|\mathcal{V}|} \frac{p_i}{n} \log_2 \left(\frac{p_i}{n}\right)$.

Ideal Diversity. The ideal diversity is the Shannon Entropy using a uniform probability distribution, implying an even distribution of nodes across all versions. $H_{ideal} = -\sum_{i=1}^{|\mathcal{V}|} \frac{1}{|\mathcal{V}|} \log_2 \left(\frac{1}{|\mathcal{V}|}\right)$.

Diversity ratio. It is useful to have a normalized diversity metric when the number of states changes, which we call the diversity ratio $d = H/H_{ideal}$.

Single point of failure. A dimension is called a *single point of failure* if there exists a version with an impact p such that $p > f$ for a distributed protocol's fault threshold f.

5.2.2 Theoretical analysis.

In this section we derive the conditions for a dimension to avoid being a single point of failure.

Firstly, we can show that by definition, a dimension is not a single point of failure if none of its versions affect more than f nodes.

LEMMA 9. *A dimension with the set $\mathcal{V}$ of versions is not a single point of failure if $p \leq f$ for all versions $v_i \in \mathcal{V}$.*

PROOF. Assume there exists a version v with impact $p > f$. If v is faulty we have more faulty nodes than the threshold f by definition. □

We can infer that as a result, for $f = \lfloor n/3 \rfloor$, $|\mathcal{V}| > 4$ must hold or at least one version has an impact $\geq n/3$. Similarly for $f = \lfloor n/2 \rfloor$, $|\mathcal{V}| > 3$ must hold. Generally, we obtain Theorem 12.

THEOREM 12. *If $|\mathcal{V}| < \lceil n/f \rceil$, for a dimension A's set $\mathcal{V}$ of versions, then A is a single point of failure.*

PROOF. We show that if $|\mathcal{V}| < \lceil n/f \rceil$, there exists a version V such that $p > f$ for its impact p.

We proceed by contradiction. Assume that if $|\mathcal{V}| < \lceil n/f \rceil$, $p_i \leq f, \forall p_i \in \mathcal{V}$ holds. We have $\sum p_i \leq |\mathcal{V}|f$ and $\sum p_i = n$. Hence $n \leq |\mathcal{V}|f$ and $|\mathcal{V}| \geq n/f$, a contradiction. □

We can now obtain sufficient conditions for a dimension to not be a single point of failure.

THEOREM 13. *If for a dimension A we have $|\mathcal{V}| \geq \lceil n/f \rceil$, for A's set $\mathcal{V}$ of versions, $H = H_{ideal}$, A is not a single point of failure.*

PROOF. We show that $p_i \leq f, \forall p_i \in \mathcal{V}$ holds if $|\mathcal{V}| \geq \lceil n/f \rceil$ and $H = H_{ideal}$ by contradiction. Assume $\exists v_j$s.t. $p_j > f$ and that $|\mathcal{V}| \geq \lceil n/f \rceil$ and $H = H_{ideal}$. We have $\sum p_i = n$ and $H = -\sum_{i=1}^{|\mathcal{V}|} \frac{p_i}{n} \log_2 \left(\frac{p_i}{n}\right)$. Since $H = H_{ideal}$, $\frac{p_i}{n} = \frac{p_j}{n}$ for all p_i and their respective version. We therefore have $n = \sum p_i = |\mathcal{V}|p_j > |\mathcal{V}|f$, implying $n > |\mathcal{V}|f$ and $|\mathcal{V}| < n/f$, a contradiction. □

5.2.3 Incentives model.

We aim to achieve ideal diversity with at least 4 versions to avoid single points of failures when $f \geq n/3 - 1$. We first describe the incentives model, simulating rational nodes maximizing individual rewards.

Node u selects a version v from the set $\mathcal{V} = \{v_1, v_2, ..., v_m, \perp\}$ of available versions by generating an attestation proof $\pi_{v,u}$ using an ideal attestation primitive. Nodes may also opt-out and select the non-version $\perp$ without any attestation proof, foregoing any rewards until a version in $\mathcal{V}$ is selected. The system runs in discrete epochs starting at epoch 0 until epoch E, where every node is initialized with $\perp$.

Each version v_i is associated with a running cost $C(v_i)$ and an allocated reward per node $R(v_i, e)$ at each epoch e. Nodes have access to the previous epoch's allocated rewards for each version.

Switch frequency and cooldown. Each node is assigned a switch threshold value τ using a uniform distribution between 0 and a switch frequency parameter $freq$. The value τ determines the threshold for which a node switches versions to obtain higher rewards.

Each node is assigned a switch cooldown value κ using a uniform distribution between 1 and the switch cooldown parameter cd. After selecting a new version, the value κ of a node determines the number of epochs where it will not change selected versions.

Rational node behavior. Every epoch e, u can switch from its current version v_i to a new version v_j if the following holds. $R(v_j, e-1) - C(v_j) > (R(v_i, e-1) - C(v_i)) \cdot (1 + \tau_u)$ and $e > e_{switched} + \kappa_u$ where $e_{switched}$ denotes the epoch where u switched versions, initialized at epoch 0.

Reward distribution. A version v_i is assigned a pool $P(v_i, e)$ of rewards at epoch e. Each node that has selected v_i at epoch e receives $R(v_i, e) = P(v_i, e)/n_{v_i,e}$ rewards where $n_{v_i,e}$ is the number of nodes that have selected v_i at epoch e.

Reaching a Nash equilibrium.

When no node in the system can increase its rewards by switching, we have reached a Nash equilibrium by definition [**103**].

In our incentives model, a Nash equilibrium is reached at epoch e when for all nodes u, their switch threshold value τ_u, and their respective selected version v_i we have $(R(v_i, e) - C(v_i) \cdot (1 + \tau_u) > R(v_j, e) - C(v_j) \forall v_j \in \mathcal{V} \setminus v$.

5.3 Incentive mechanisms.

5.3.1 PID controller

A Proportional-Integral-Derivative (PID) controller is a closed-loop control mechanism that aims to minimize the error between an observed system output and a desired target value

through continuous adjustment of control inputs [**14**, **59**]. In the context of blockchain client diversity, we formalize the PID control problem as follows.

System formulation. Let $r_v(e) = \frac{p(e)}{n}$ denote the observed fraction of nodes running version $v \in \mathcal{V}$ at epoch e, where $p(e)$ is the number of nodes running version v.

The target fraction for ideal diversity is $t = \frac{1}{|\mathcal{V}|}$

The error signal for version v at epoch e is defined as:

$$\epsilon_v(e) = t - r_v(e) = \frac{1}{|\mathcal{V}|} - \frac{n_v(e)}{n}$$

PID control equations [17]. The PID controller output for version v consists of three components. In our epoch-based system the components are as follows:

- Proportional $P_v(e) = K_P \cdot \epsilon_v(e)$
- Integral $I_v(e) = K_I \cdot \sum_{i=0}^{e} \epsilon_v(i)$
- Derivative $D_v(e) = K_D \cdot [\epsilon_v(e) - \epsilon_v(e-1)]$

The total control output, which determines the reward allocation for version v at epoch $e+1$, is:

$$R_v(e+1) = P_v(e) + I_v(e) + D_v(e)$$

where K_P, K_I, and K_D are the proportional, integral, and derivative parameters, respectively.

The proportional term provides immediate response proportional to the current error. The integral term eliminates steady-state error by accumulating past errors. The derivative term provides anticipatory control by responding to the rate of error change, improving system stability and reducing overshoot.

Determining proper values for the PID parameters K_P, K_I, and K_D is not trivial, and is commonly done by trial-and-error in industrial control systems. For our rewards system, a systematic approach is desired to avoid trial-and-error tuning.

5.3.2 Ziegler-Nichols (ZN) tuning.

The Ziegler-Nichols tuning method provides a systematic approach to determine optimal PID parameters through experimental characterization of the system's dynamic response [**134**]. This classical method eliminates manual trial-and-error while ensuring good performance, making it one of the most widely adopted tuning techniques in industrial control applications [**43**, **105**].

The ZN tuning method initially sets K_I and K_D to 0, then gradually increases K_P from a starting value until oscillations are detected in the system. Once oscillations are detected, the corresponding K_P value is saved as the ultimate gain and the oscillation period is saved as the ultimate period.

Oscillation detection criteria. To detect these oscillations robustly, we employ the Fast Fourier Transform (FFT) method [**126**, **127**]. The FFT on a window of W epochs outputs the corresponding observed frequencies and their amplitudes.

Sustained oscillations are detected when the amplitude of a frequency is over $0.1 \cdot n$ and the period T_u is within the range $2 < T_u < W/2$.

Automated tuning procedure. The complete Ziegler-Nichols procedure is given in Appendix **??**.

5.3.3 Ziegler-Nichols tuned PID controller algorithm

The Ziegler-Nichols tuned PID controller implementation is specified in Algoritm 4, and represented in Figure 5.1.

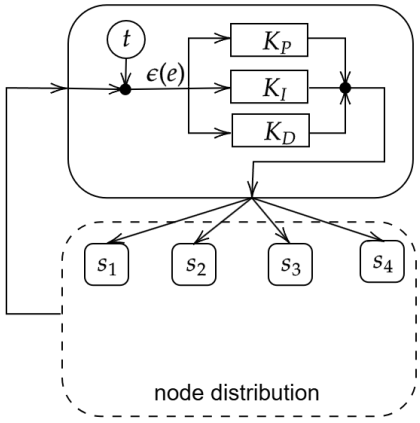


FIGURE 5.1. Diagram representing our fixed tuning PID-based system.

Algorithm 4 Complete Ziegler-Nichols PID System

Initialize: error_sum$_v = 0$, error_prev$_v = 0$ for all $v \in \mathcal{V}$
Parameters: $K_P^{(0)}$, ΔK, observation window W, total epochs E
// Tuning phase
$(K_P, K_I, K_D) \leftarrow$ ZieglerNicholsTuning$(K_P^{(0)}, \Delta K, W)$
// Operational control
for epoch $e = 0$ to E **do**
  Observe current node distribution $\{p(e)\}_{v \in \mathcal{V}}$
  Distribute rewards $R(v, e) \forall v \in \mathcal{V}$
  for each version $v \in \mathcal{V}$ **do**
    Compute current error: $\epsilon_v(e) = \frac{1}{|\mathcal{V}|} - \frac{p(e)}{n}$
    // Update accumulated error
    error_sum$_v \leftarrow$ error_sum$_v + \epsilon_v(e)$
    // Compute last error
    error_deriv$_v \leftarrow \epsilon_v(e) -$ error_prev$_v$
    // Update state for next iteration
    error_prev$_v \leftarrow \epsilon_v(e)$
    // Compute PID components
    $P_v \leftarrow K_P \cdot \epsilon_v(e)$
    $I_v \leftarrow K_I \cdot$ error_sum$_v$
    $D_v \leftarrow K_D \cdot$ error_deriv$_v$
    // Compute control output
    $R(v, e+1) \leftarrow \max(0, P_v + I_v + D_v)$
  end for
end for

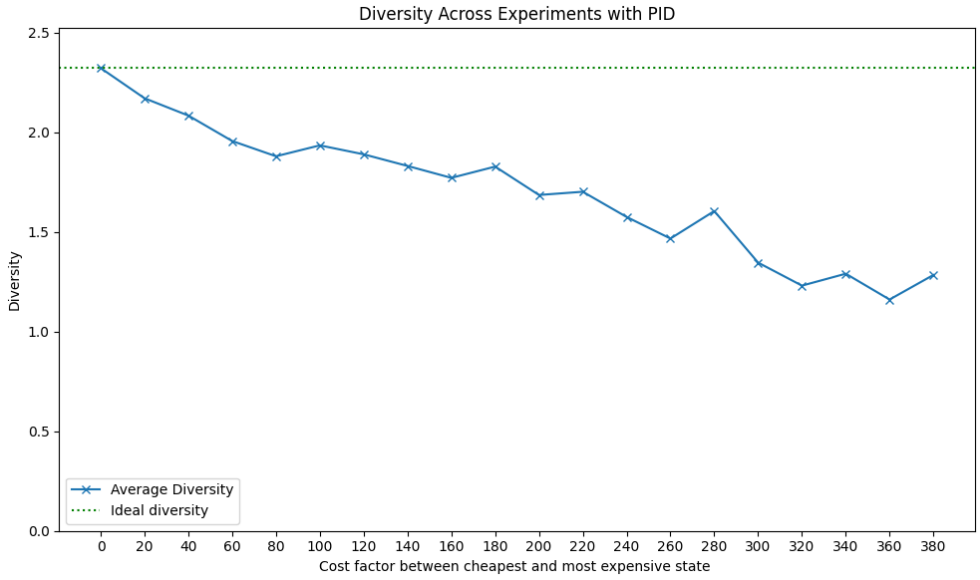


FIGURE 5.2. Diversity at the end of 1000 epochs with a growing factor in running costs between the cheapest and most expensive version.

5.3.4 Unknown environment tuning point problem.

We are able to obtain good performance using the Ziegler-Nichols tuning method in our simulations, without knowledge of the real costs involved in the system, solving the unknown cost problem.

However, the obtained PID tuning is still specific to the simulated environment's parameters such as real costs, and node switching frequency. It is therefore not obvious that a set of PID parameters obtained in one specific environment still performs well if the environment changes. To evaluate the PID controller that is tuned for a specific environment, we vary the real costs without changing the tuning, and observe the obtained performance.

As seen in Figure 5.2, the PID controller with fixed tuning is not able to maintain good diversity as the gap between the cheapest version and the most expensive version grows. While we can observe a gradual decrease, the diversity penalty at a $20x$ factor in version costs is still close to ideal, and it could be argued that maintaining diversity for higher cost gaps is unnecessary for real deployments.

To avoid the dependence on the tuning obtained from a specific environment's parameters, the PID tuning should be able to adapt to a changing environment.

5.3.5 Reinforcement-learning tuned PID

We propose tuning the PID parameters using reinforcement learning, as explored in previous work for traditional industrial control [**41**, **52**]. Reinforcement-learning based tuning of the PID parameters hypothetically allows the system to adapt to different real system dynamics.

As our RL agent takes the continuous version distribution as input, RL agents that compute every possible version are not suited. We therefore use a double deep-Q-network (DQN) approach [**98**, **125**] with a neural network architecture to observe the continuous inputs.

Our RL agent follows established RL principles: 2-hidden layers provide universal approximation capabilities [**78**] with sufficient depth for complex control mappings, hidden layer sizes of 64 follow common practice in deep RL [**98**], ReLU activations mitigate vanishing gradients [**101**], while tanh outputs ensure bounded actions for system stability [**70**]. We employ Double DQN [**125**] with experience replay [**98**] to address overestimation bias and sample efficiency.

Our RL-tuned PID reward allocation mechanism is depicted in Figure 5.3.

State and action spaces. We define the observation space $\mathcal{O} \subset \mathbb{R}^{|\mathcal{V}|+4}$ and action space $\mathcal{A} = [-1, 1]^4$ for the RL agent. The bounded action space ensures stable parameter updates.

Observation space. The observation vector $\mathbf{o}_e \in \mathbb{R}^{|\mathcal{V}|+4}$ is constructed as:

$$\mathbf{o}_e = [\mathbf{d}_e, \tilde{\theta}_P, \tilde{\theta}_I, \tilde{\theta}_D, \tilde{r}_{\text{scale}}]^T$$

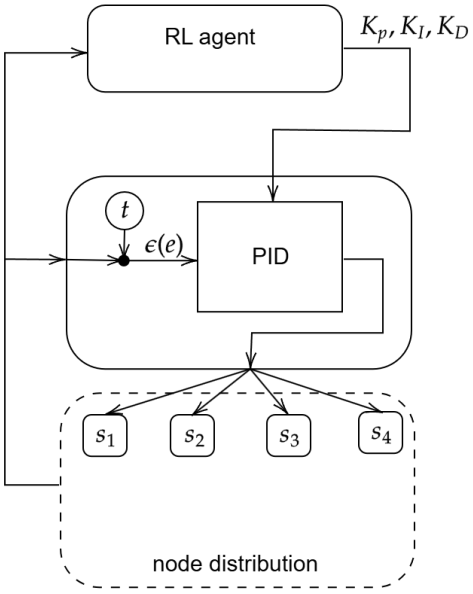


FIGURE 5.3. Diagram representing our RL-tuned PID-based system.

where:

$$\mathbf{d}_e = [n_1/n, \ldots, n_{|\mathcal{V}|}/n]^T$$

$$\tilde{\theta}_P = \frac{\log(1 + K_P)}{\log(1 + K_{P,\max})} \in [0, 1]$$

$$\tilde{\theta}_I = \frac{\log(1 + K_I)}{\log(1 + K_{I,\max})} \in [0, 1]$$

$$\tilde{\theta}_D = \frac{\log(1 + K_D)}{\log(1 + K_{D,\max})} \in [0, 1]$$

$$\tilde{r}_{\text{scale}} = \frac{\log(1 + r_{\text{scale}})}{\log(1 + r_{\text{scale,max}})} \in [0, 1]$$

The PID parameters (K_P, K_I, K_D) are the proportional, integral, and derivative gains respectively, while r_{scale} is the overall reward scaling factor. All inputs are log-normalized to $[0, 1]$ for neural network stability.

Neural network architecture. The Q-network $Q_\theta : \mathcal{O} \times \mathcal{A} \to \mathbb{R}$ maps state-action pairs to Q-values. The network architecture consists of:

$$Q_\theta(\mathbf{o}, \mathbf{a}) = W_{\text{out}} \cdot \text{ReLU}(W_2 \cdot \text{ReLU}(W_1 \cdot [\mathbf{o}; \mathbf{a}] + \mathbf{b}_1) + \mathbf{b}_2) + b_{\text{out}}$$

where $[\mathbf{o}; \mathbf{a}]$ denotes concatenation of observation and action vectors, $W_1 \in \mathbb{R}^{64 \times (|\mathcal{V}|+8)}$, $W_2 \in \mathbb{R}^{64 \times 64}$, and $W_{\text{out}} \in \mathbb{R}^{1 \times 64}$. The Rectified Linear Unit (ReLU) is defined as $\text{ReLU}(x) = \max(0, x)$.

Action selection. During training, we use ϵ-greedy exploration:

$$\mathbf{a}_e = \begin{cases} \text{random action from } \mathcal{A} & \text{if random}() < \epsilon_e \\ \arg\max_{\mathbf{a} \in \mathcal{A}} Q_\theta(\mathbf{o}_e, \mathbf{a}) & \text{otherwise} \end{cases}$$

Parameter update mechanism. Given action $\mathbf{a}_e = [\delta_P, \delta_I, \delta_D, \delta_{\text{scale}}]^T \in [-1, 1]^4$, parameters are updated as:

$$\begin{aligned} K_P^{(e+1)} &= \max(0, K_P^{(e)} + \alpha \cdot \delta_P) \\ K_I^{(e+1)} &= \max(0, K_I^{(e)} + \alpha \cdot \delta_I) \\ K_D^{(e+1)} &= \max(0, K_D^{(e)} + \alpha \cdot \delta_D) \\ r_{\text{scale}}^{(e+1)} &= \max(\epsilon_{\min}, r_{\text{scale}}^{(e)} + \alpha \cdot \delta_{\text{scale}}) \end{aligned}$$

where $\alpha > 0$ is the parameter update learning rate.

RL Reward function. The reward is defined with the diversity $H(e)$ at epoch e.

$$r(e) = \min\left(\frac{H(e)}{H_{\text{ideal}}}, 1.0\right)$$

Loss function. We use the Double DQN loss function to train the Q-network:

$$L(\theta) = \mathbb{E}_{(\mathbf{o}, \mathbf{a}, r, \mathbf{o}') \sim \mathcal{D}} \left[(y - Q_\theta(\mathbf{o}, \mathbf{a}))^2 \right]$$

where the target y is computed using the Double DQN update rule:

$$y = r + \gamma \cdot Q_{\theta^-}\left(\mathbf{o}', \arg\max_{\mathbf{a}'} Q_\theta(\mathbf{o}', \mathbf{a}')\right)$$

Here, Q_{θ^-} is the target network with parameters θ^-, updated periodically as $\theta^- \leftarrow \theta$, and $\gamma \in [0, 1]$ is the discount factor.

Training algorithm. The pseudocode for the RL agent algorithm between epochs 0 and E is given in Algorithm 5.

Algorithm 5 RL-tuned PID training

Initialize: Q-network Q_θ, target network Q_{θ^-}, replay buffer $\mathcal{D} = \emptyset$
Parameters: $\epsilon_0, \epsilon_{\min}, \epsilon_{\text{decay}}, \alpha, \gamma$, batch_size, update_freq, target_update_freq
for epoch $e = 0$ to E **do**
  Observe current $\mathbf{o}_e$
  if random() $< \epsilon_e$ **then**
    $\mathbf{a}_e \leftarrow$ random action from $\mathcal{A} = [-1, 1]^4$
  else
    $\mathbf{a}_e \leftarrow \arg\max_{\mathbf{a} \in \mathcal{A}} Q_\theta(\mathbf{o}_e, \mathbf{a})$
  end if
  Apply PID parameter updates using $\mathbf{a}_e$
  Execute PID control to generate reward allocation $\mathbf{R}_e$
  Distribute rewards and observe $\mathbf{o}_{e+1}$
  Compute immediate reward $r(e)$
  Store transition $(\mathbf{o}_e, \mathbf{a}_e, r(e), \mathbf{o}_{e+1})$ in $\mathcal{D}$
  if $|\mathcal{D}| \geq$ batch_size **and** $e \bmod$ update_freq $= 0$ **then**
    Sample $\mathcal{B} = \{(\mathbf{o}_i, \mathbf{a}_i, r_i, \mathbf{o}'_i)\}_{i=1}^{\text{batch\_size}}$ from $\mathcal{D}$
    Compute target values $y_i = r_i + \gamma \cdot Q_{\theta^-}(\mathbf{o}'_i, \arg\max_{\mathbf{a}'} Q_\theta(\mathbf{o}'_i, \mathbf{a}'))$
    Update Q_θ by minimizing loss $L(\theta) = \frac{1}{|\mathcal{B}|} \sum_i (y_i - Q_\theta(\mathbf{o}_i, \mathbf{a}_i))^2$
  end if
  if $e \bmod$ target_update_freq $= 0$ **then**
    $\theta^- \leftarrow \theta$
  end if
  $\epsilon_e \leftarrow \max(\epsilon_{\min}, \epsilon_e \cdot \epsilon_{\text{decay}})$
end for

Guiding values. It may be possible to achieve better results by guiding the RL agent with quantitative values, especially the scaling factor for proper PID parameters. However, due

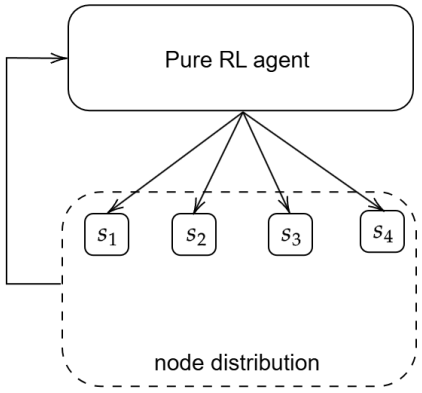


FIGURE 5.4. Diagram representing our Pure RL based system.

to the lack of real-world data for our use-case, we abstained from any environment-specific values in our model.

5.3.6 Pure reinforcement learning method.

In our pure reinforcement learning method, the RL agent directly allocates rewards for each version, depicted in Figure 5.4. As a result, the action space grows with the number of versions. We use the same established principles as for the RL-tuned PID method. To accommodate for the larger action space, we expand the deep network width to 128 as generally recommended for more complex tasks [**98**].

Initially, we obtained only poor results using this method, and therefore focused on the PID-based methods. However, by limiting the action space such that negative allocated rewards are not allowed, the system is able to converge. In addition, to maximize its diversity-based reward, we struggled with large spikes in increased total allocated rewards to obtain slightly better diversity. We therefore suspend all RL actions when the diversity ratio is close to 1 to prevent unbounded total rewards being allocated.

As the approach directly operates on each version's rewards, we can also gradually reduce all allocated rewards when the diversity ratio is above 0.995. In our case, we used the conservative 0.1% reward reduction every epoch when above 0.995 in diversity.

Observation space: The observation vector for the direct reward allocation controller is defined as:

$$\mathbf{o}'_e \in \mathbb{R}^{2|\mathcal{V}|+1}$$

where the observation is constructed as:

$$\mathbf{o}'_e = [\mathbf{d}_e, \tilde{\mathbf{R}}_e, \tilde{r}_{\max}]^T$$

with components:

$$\begin{aligned}
\mathbf{d}_e &= [n_1/n, n_2/n, \ldots, n_{|\mathcal{V}|}/n]^T \in \mathbb{R}^{|\mathcal{V}|} \\
\tilde{\mathbf{R}}_e &= [R_1/r_{\max}, R_2/r_{\max}, \ldots, R_{|\mathcal{V}|}/r_{\max}]^T \in [0,1]^{|\mathcal{V}|} \\
\tilde{r}_{\max} &= \frac{r_{\max}}{r_{\text{initial}} \cdot \xi} \in [0,1]
\end{aligned}$$

where n_i represents the number of agents in version i, R_i is the current reward for version i, $r_{\max}$ is the current maximum reward bound, r_{initial} is the initial maximum reward, and $\xi = 10$ allows for reward bound expansion.

Action space: The action space is version-dependent:

$$\mathcal{A}' = [-1, 1]^{|\mathcal{V}|+1}$$

where the action vector is:

$$\mathbf{a}'_e = [a_1, a_2, \ldots, a_{|\mathcal{V}|}, a_{\max}]^T$$

with a_i representing reward adjustment for version i and $a_{\max}$ controlling the maximum reward bound.

Neural network architecture: The Q-network $Q_\phi : \mathbb{R}^{2|\mathcal{V}|+1} \times \mathcal{A}' \to \mathbb{R}$ maps state-action pairs to Q-values, where ϕ denotes the network parameters. The network architecture consists of:

$$Q_\phi(\mathbf{o}', \mathbf{a}') = W_{\text{out}} \cdot \text{ReLU}(W_2 \cdot \text{ReLU}(W_1 \cdot [\mathbf{o}'; \mathbf{a}'] + \mathbf{b}_1) + \mathbf{b}_2) + b_{\text{out}}$$

where $[\mathbf{o}'; \mathbf{a}']$ denotes concatenation of observation and action vectors, and:

$$W_1 \in \mathbb{R}^{128 \times (3|\mathcal{V}|+2)} \quad \text{(input layer)}$$

$$W_2 \in \mathbb{R}^{128 \times 128} \quad \text{(hidden layer)}$$

$$W_{\text{out}} \in \mathbb{R}^{1 \times 128} \quad \text{(output layer)}$$

Action selection. During training, we use ϵ-greedy exploration:

$$\mathbf{a}'_e = \begin{cases} \text{random action from } \mathcal{A}' & \text{if random}() < \epsilon_e \\ \arg\max_{\mathbf{a}' \in \mathcal{A}'} Q_\phi(\mathbf{o}'_e, \mathbf{a}') & \text{otherwise} \end{cases}$$

Reward allocation mechanism: Given action $\mathbf{a}'_e = [a_1, a_2, \ldots, a_{|\mathcal{V}|}, a_{\max}]^T \in [-1, 1]^{|\mathcal{V}|+1}$, the pool $P(v_i, e+1)$ of rewards for version v_i is given as:

$$P(v_i, e+1) = r_{\min} + \frac{a_i + 1}{2} \cdot (r_{\max}^{(e+1)} - r_{\min})$$

where $r_{\min} = 0.1 \cdot r_{\max}^{(e+1)}$ ensures minimum reward provision, and:

$$r_{\max}^{(e+1)} = \max(r_{\min,\text{global}}, r_{\max}^{(e)} + \alpha \cdot a_{\max} \cdot r_{\text{initial}})$$

with $r_{\min,\text{global}} = 0.1 \cdot r_{\text{initial}}$ preventing the agent from collapsing to 0 rewards, causing all nodes to stay on version $\perp$, and $\alpha > 0$ is the reward adjustment learning rate.

For version $\perp$, the allocated reward is always set to zero:

RL Reward function: The immediate RL reward is defined as for the RL-tuned PID method.

Stability mechanism: When diversity approaches optimality ($H(e)/H_{\text{ideal}} > 0.995$), the system applies conservative reward reduction:

$$R_i^{(e+1)} = R_i^{(e)} \cdot (1 - \lambda) \quad \forall i \neq \perp$$

where $\lambda = 0.001$ provides gentle reduction while maintaining relative reward proportions.

Loss function. We use the Double DQN loss function to train the Q-network:

$$L(\phi) = \mathbb{E}_{(\mathbf{o}', \mathbf{a}', r', \mathbf{o}'') \sim \mathcal{D}'} \left[\left(y - Q_\phi(\mathbf{o}', \mathbf{a}') \right)^2 \right]$$

where the target y is computed using the Double DQN update rule:

$$y = r' + \gamma \cdot Q_{\phi^-} \left(\mathbf{o}'', \arg\max_{\mathbf{a}''} Q_\phi(\mathbf{o}'', \mathbf{a}'') \right)$$

Here, Q_{ϕ^-} is the target network with parameters ϕ^-, updated periodically as $\phi^- \leftarrow \phi$, and $\gamma \in [0, 1]$ is the discount factor.

Training algorithm. The pseudocode for the RL agent algorithm between epochs 0 and E is given in Algorithm 6.

Algorithm 6 Pure RL Training

Initialize: Q-network Q_ϕ, target network Q_{ϕ^-}, replay buffer $\mathcal{D}' = \emptyset$
Parameters: $\epsilon_0, \epsilon_{\min}, \epsilon_{\text{decay}}, \alpha, \gamma$, batch_size, update_freq, target_update_freq
for epoch $e = 0$ to E **do**
  Observe current observation $\mathbf{o}'_e$
  Compute current diversity $H(e)$ and diversity ratio $\rho_e = H(e)/H_{\text{ideal}}$
  if $\rho_e > 0.995$ **then**
    Apply stability mechanism: $R_v^{(e+1)} \leftarrow R_v^{(e)} \cdot (1 - \lambda)$ for $v \neq \perp$
  else
    if random() $< \epsilon_e$ **then**
      $\mathbf{a}'_e \leftarrow$ random action from $\mathcal{A}' = [-1, 1]^{|\mathcal{V}|+1}$
    else
      $\mathbf{a}'_e \leftarrow \arg\max_{\mathbf{a}' \in \mathcal{A}'} Q_\phi(\mathbf{o}'_e, \mathbf{a}')$
    end if
    Apply reward allocation using $\mathbf{a}'_e$
    Execute reward distribution to agents and observe next observation $\mathbf{o}'_{e+1}$
    Compute immediate reward r'_e
    Store transition $(\mathbf{o}'_e, \mathbf{a}'_e, r'_e, \mathbf{o}'_{e+1})$ in $\mathcal{D}'$
    if $|\mathcal{D}'| \geq$ batch_size **and** $e \bmod$ update_freq $= 0$ **then**
      Sample $\mathcal{B}' = \{(\mathbf{o}'_i, \mathbf{a}'_i, r'_i, \mathbf{o}''_i)\}_{i=1}^{\text{batch\_size}}$ from $\mathcal{D}'$
      Compute target values $y_i = r'_i + \gamma \cdot Q_{\phi^-}(\mathbf{o}''_i, \arg\max_{\mathbf{a}''} Q_\phi(\mathbf{o}''_i, \mathbf{a}''))$
      Update Q_ϕ by minimizing loss $L(\phi) = \frac{1}{|\mathcal{B}'|} \sum_i (y_i - Q_\phi(\mathbf{o}'_i, \mathbf{a}'_i))^2$
    end if
    if $e \bmod$ target_update_freq $= 0$ **then**
      $\phi^- \leftarrow \phi$
    end if
  end if
  $\epsilon_e \leftarrow \max(\epsilon_{\min}, \epsilon_e \cdot \epsilon_{\text{decay}})$
end for

5.3.7 Dynamic version addition for RL controllers

When a new version v_{new} is introduced to the system at epoch e^*, both RL controllers must adapt their architectures for the new set $V' = V \cup v_{\text{new}}$ of versions

5.3.7.1 RL-tuned PID controller adaptation

For the RL-tuned PID controller, the introduction of a new version affects only the observation space dimension, while the action space remains fixed.

Observation space expansion: The observation space expands from $\mathbb{R}^{|\mathcal{V}|+4}$ to $\mathbb{R}^{|\mathcal{V}|+5}$:

$$\mathbf{o}_{e,\text{old}} = [\mathbf{d}_{e,\text{old}}, \tilde{\theta}_P, \tilde{\theta}_I, \tilde{\theta}_D, \tilde{r}_{\text{scale}}]^E \in \mathbb{R}^{|\mathcal{V}|+4}$$
$$\mathbf{o}_{e,\text{new}} = [\mathbf{d}_{e,\text{new}}, \tilde{\theta}_P, \tilde{\theta}_I, \tilde{\theta}_D, \tilde{r}_{\text{scale}}]^E \in \mathbb{R}^{|\mathcal{V}|+5}$$

where $\mathbf{d}_{e,\text{new}} = [n_1/n, \ldots, n_{|\mathcal{V}|}/n, n_{\text{new}}/n]^E$ includes the new version's agent distribution.

Network weight transfer: The input layer weight matrix $W_1 \in \mathbb{R}^{64\times(|\mathcal{V}|+4)}$ is expanded to $W_1' \in \mathbb{R}^{64\times(|\mathcal{V}|+5)}$ by copying the existing weights, initializing the new version's weight with a normal distribution, and shifting the PID parameter weights by one position in the matrix accordingly. More formally we have:

$$W_1'[i,j] = \begin{cases} W_1[i,j] & \text{if } j \leq |\mathcal{V}| \\ \mathcal{N}(0, \sigma_{\text{init}}^2) & \text{if } j = |\mathcal{V}|+1 \\ W_1[i,j-1] & \text{if } j > |\mathcal{V}|+1 \end{cases}$$

where $\sigma_{\text{init}} = 0.1$ provides conservative initialization.

PID error scaling: The accumulated and last error terms for existing versions are scaled to account for the changed ideal distribution. Otherwise, the new ideal diversity causes previous error measurements to be inaccurate.

$$\text{ideal\_share}_{\text{old}} = \frac{1}{|\mathcal{V}|}$$
$$\text{ideal\_share}_{\text{new}} = \frac{1}{|\mathcal{V}|+1}$$
$$\text{scale\_factor} = \frac{\text{ideal\_share}_{\text{old}}}{\text{ideal\_share}_{\text{new}}} = \frac{|\mathcal{V}|}{|\mathcal{V}|+1}$$

For each existing version $v \in \mathcal{V}$:

$$\text{error}_{\text{accumulated}}[v] \leftarrow \text{error}_{\text{accumulated}}[v] \times \text{scale\_factor}$$

$$\text{error}_{\text{last}}[v] \leftarrow \text{error}_{\text{last}}[v] \times \text{scale\_factor}$$

5.3.7.2 Direct reward allocation controller adaptation

The direct allocation controller faces more complex adaptation due to changes in both observation and action spaces.

Observation space expansion: The observation space grows from $\mathbb{R}^{2|\mathcal{V}|+1}$ to $\mathbb{R}^{2(|\mathcal{V}|+1)+1}$:

$$\mathbf{o}'_{e,\text{old}} = [\mathbf{d}_{e,\text{old}}, \tilde{\mathbf{R}}_{e,\text{old}}, \tilde{r}_{\max}]^E \in \mathbb{R}^{2|\mathcal{V}|+1}$$

$$\mathbf{o}'_{e,\text{new}} = [\mathbf{d}_{e,\text{new}}, \tilde{\mathbf{R}}_{e,\text{new}}, \tilde{r}_{\max}]^E \in \mathbb{R}^{2|\mathcal{V}|+3}$$

where $\tilde{\mathbf{R}}_{e,\text{new}}$ includes the normalized reward for the new version.

Action space expansion: The action space expands from $[-1, 1]^{|\mathcal{V}|+1}$ to $[-1, 1]^{|\mathcal{V}|+2}$:

$$\mathbf{a}'_{e,\text{old}} = [a_1, \ldots, a_{|\mathcal{V}|}, a_{\max}]^E$$

$$\mathbf{a}'_{e,\text{new}} = [a_1, \ldots, a_{|\mathcal{V}|}, a_{\text{new}}, a_{\max}]^E$$

Input layer adaptation: The input layer weights $W_1 \in \mathbb{R}^{128\times(2|\mathcal{V}|+1)}$ are expanded to $W'_1 \in \mathbb{R}^{128\times(2|\mathcal{V}|+3)}$ by copying the existing node distribution and version reward weights, and initializing the new version's node distribution and reward weights with a normal distribution. More formally we have:

$$W_1'[i,j] = \begin{cases} W_1[i,j] & \text{if } j \leq |\mathcal{V}| \\ \mathcal{N}(0, \sigma_{\text{init}}^2) & \text{if } j = |\mathcal{V}| + 1 \\ W_1[i, j-1] & \text{if } |\mathcal{V}| + 2 \leq j \leq 2|\mathcal{V}| + 1 \\ \mathcal{N}(0, \sigma_{\text{init}}^2) & \text{if } j = 2|\mathcal{V}| + 2 \\ W_1[i, 2|\mathcal{V}| + 1] & \text{if } j = 2|\mathcal{V}| + 3 \end{cases}$$

Output layer adaptation: The output layer weights $W_3 \in \mathbb{R}^{(|\mathcal{V}|+1)\times 128}$ are expanded to $W_3' \in \mathbb{R}^{(|\mathcal{V}|+2)\times 128}$ by copying the existing weights, initializing the new version's weight with a normal distribution, and shifting the maximum reward parameter weight by one position in the matrix accordingly. More formally we have:

$$W_3'[i,j] = \begin{cases} W_3[i,j] & \text{if } i \leq |\mathcal{V}| \\ \mathcal{N}(0, \sigma_{\text{init}}^2) & \text{if } i = |\mathcal{V}| + 1 \\ W_3[|\mathcal{V}| + 1, j] & \text{if } i = |\mathcal{V}| + 2 \end{cases}$$

5.3.7.3 Common adaptation mechanisms

Both controllers employ shared stabilization mechanisms during the adaptation period:

Exploration reset: The exploration rate is reset to encourage discovery in the expanded space:

Transition protection period: Learning is briefly suspended to allow nodes to adjust to the added version.

Initial reward assignment: The new version is initialized with zero reward allocation $R_{\text{new}}^{(e^*)} = 0$.

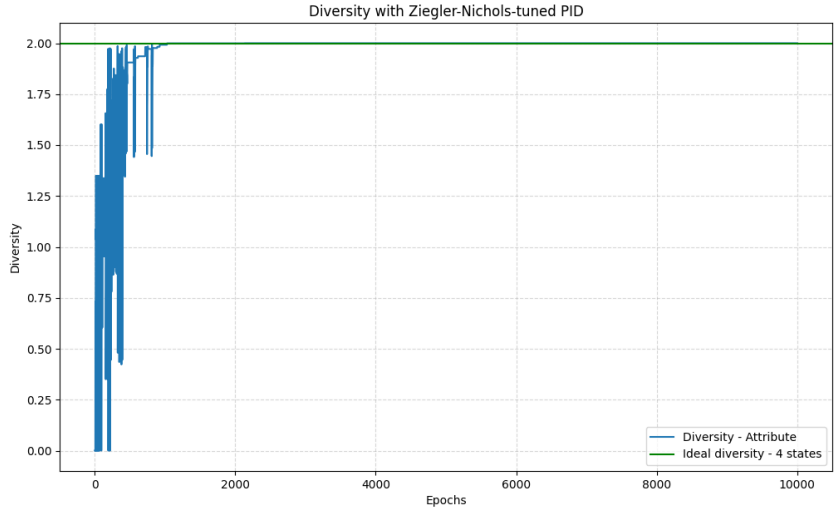


FIGURE 5.5. Simulated diversity using Ziegler-Nichols-tuned PID over 10000 epochs for 4 states and $n = 100$ nodes. The switch frequency parameter is 0.5 and the switch cooldown is between 1 and 10.

5.4 Simulation results

5.4.1 PID tuned with Ziegler-Nichols.

The PID tuned with the Ziegler-Nichols method is able to make the system converge to the ideal diversity as observed in Figure 5.5 and visualized in Figure 5.6. The obtained PID parameters by Ziegler-Nichols tuning in this experiment are: $K_P = 573.8$, $K_I = 247.9$, and $K_D = 332.0$.

The allocated rewards stabilize when nodes have converged, as seen in Figure 5.7 and Figure 5.8.

5.4.2 PID tuned with reinforcement learning.

For the PID tuned with RL, we observe convergence after around 1500 epochs in Figure 5.9 and visualized in Figure 5.10. We also confirm that this method adapts to a changing cost

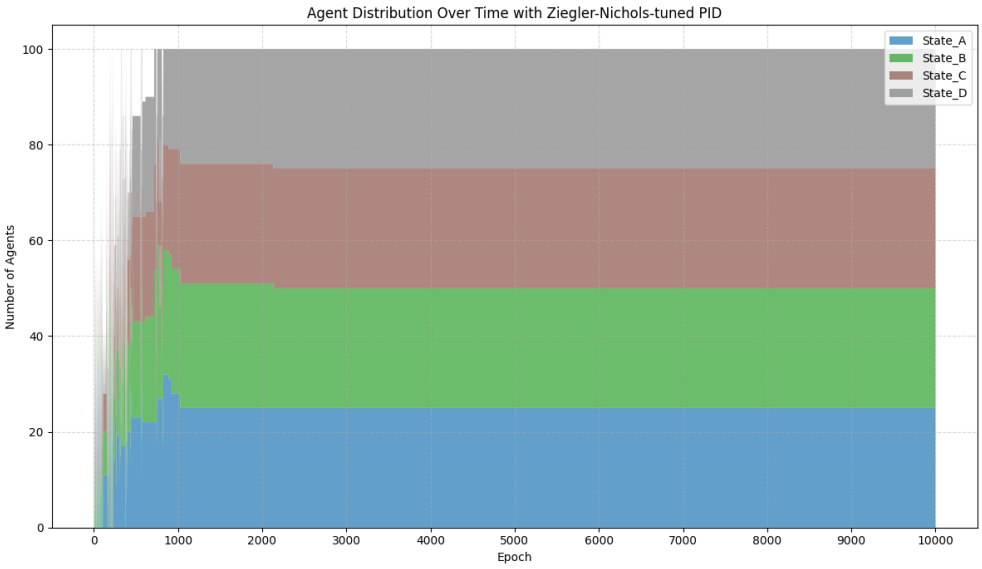


FIGURE 5.6. Node allocation per state using Ziegler-Nichols-tuned PID over 10000 epochs for 4 states and $n = 100$ nodes. The switch frequency parameter is 0.5 and the switch cooldown is between 1 and 10.

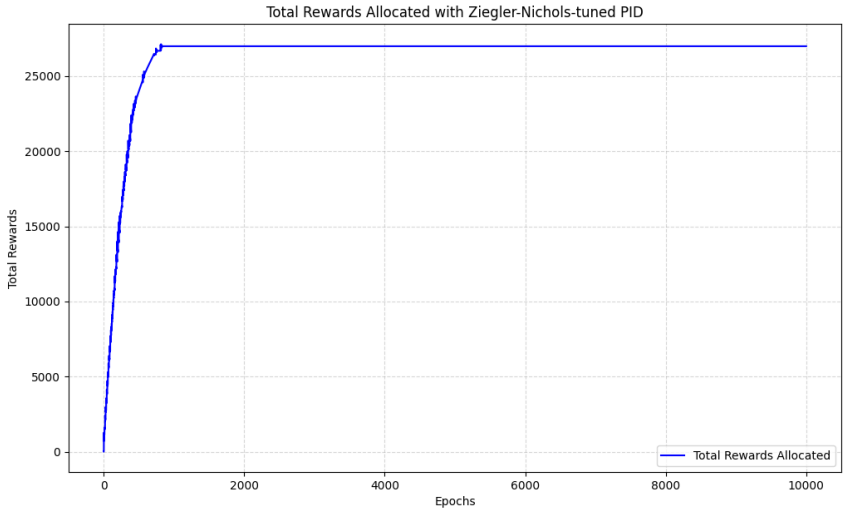


FIGURE 5.7. Total rewards using Ziegler-Nichols-tuned PID over 10000 epochs for 4 states and $n = 100$ nodes. The switch frequency parameter is 0.5 and the switch cooldown is between 1 and 10.

distribution better than when tuned once with the Ziegler-Nichols method, reaching ideal diversity even when the running cost gap is $100x$ in Figure 5.11.

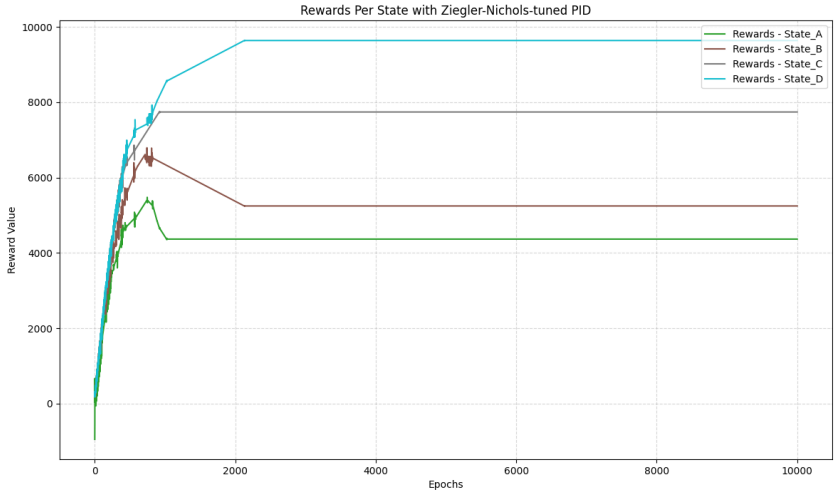


FIGURE 5.8. Rewards per state using Ziegler-Nichols-tuned PID over 10000 epochs for 4 states and $n = 100$ nodes. The switch frequency parameter is 0.5 and the switch cooldown is between 1 and 10.

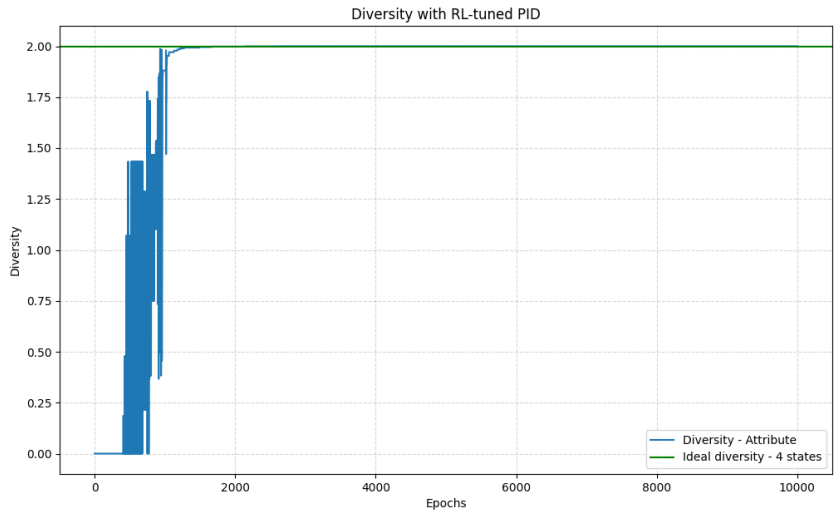


FIGURE 5.9. Simulated diversity using RL-tuned PID over 10000 epochs for 4 states and $n = 100$ nodes. The switch frequency parameter is 0.5 and the switch cooldown is between 1 and 10.

The allocated rewards stabilize when nodes have converged, as seen in Figure 5.12 and Figure 5.13.

FIGURE 5.10. Node allocation per state using RL-tuned PID over 10000 epochs for 4 states and $n = 100$ nodes. A new state is added at epoch 5000. The switch frequency parameter is 0.5 and the switch cooldown is between 1 and 10.

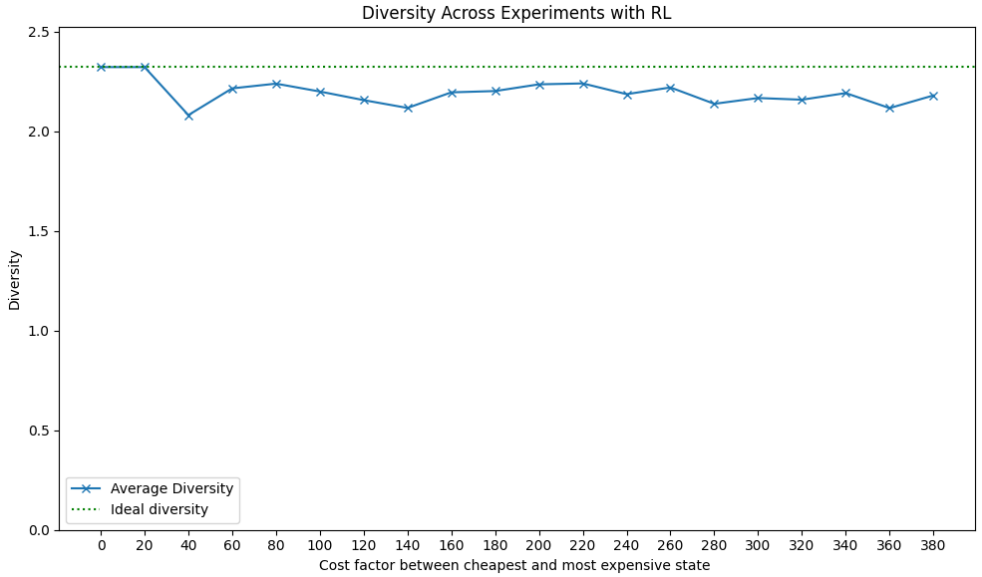


FIGURE 5.11. Diversity using RL-tuned PID at the end of 1000 epochs with a growing factor in running costs between the cheapest and most expensive state..

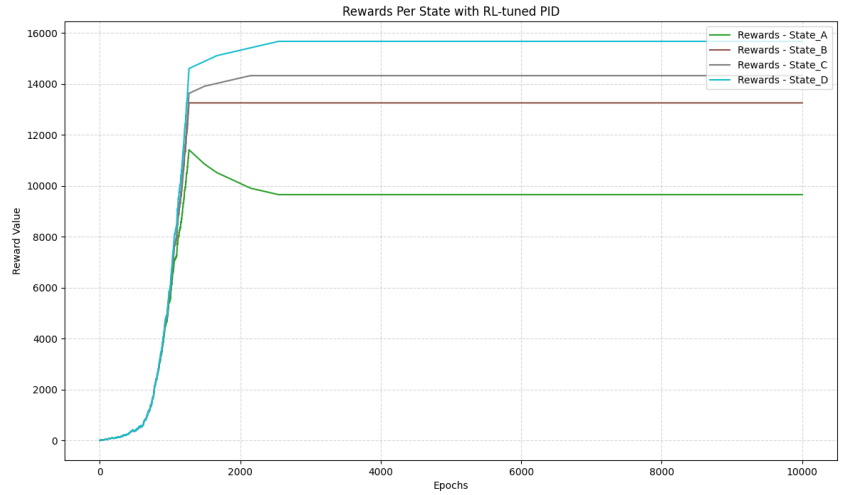


FIGURE 5.12. Rewards per state using RL-tuned PID over 10000 epochs for 4 states and $n = 100$ nodes. The switch frequency parameter is 0.5 and the switch cooldown is between 1 and 10.

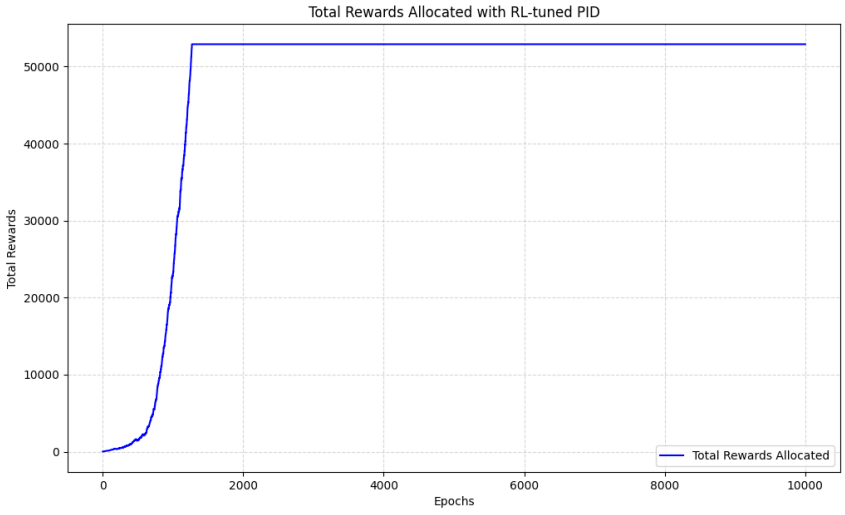


FIGURE 5.13. Total rewards using RL-tuned PID over 10000 epochs for 4 states and $n = 100$ nodes. The switch frequency parameter is 0.5 and the switch cooldown is between 1 and 10.

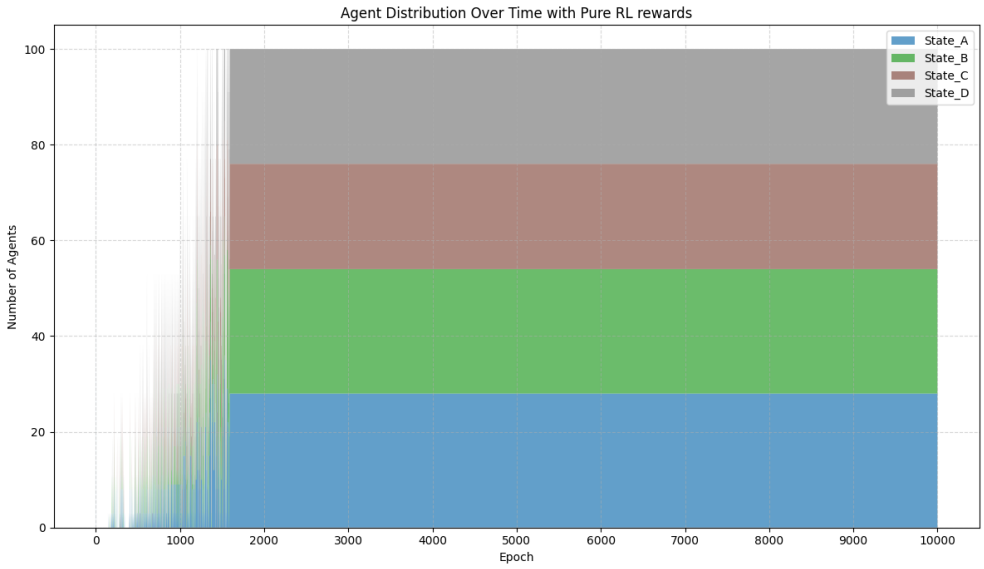


FIGURE 5.14. Node assignments of pure RL over 10000 epochs for 4 states and $n = 100$ nodes. The switch frequency parameter is 0.5 and the switch cooldown is between 1 and 10.

5.4.3 Pure RL method.

Our pure RL method has a longer training phase, converging to ideal diversity only after 1800 epochs as observed in Figure 5.15 and visualized in Figure 5.14.

As for the RL-tuned PID, we confirm that this method adapts to a changing cost distribution better than the Ziegler-Nichols method, reaching ideal diversity even when the running cost gap is 100% in Figure 5.16.

The allocated rewards rise rapidly before stabilizing when nodes have converged, as seen in Figure 5.18 and Figure 5.17.

When gently decreasing total rewards at a rate of 0.01% per epoch, we observe sudden instability at stages where rewards become too low in Figure 5.19. The large swing in diversity suggest that the lower rewards observed in Figure 5.20 may not be worth the instability in Figure 5.19.

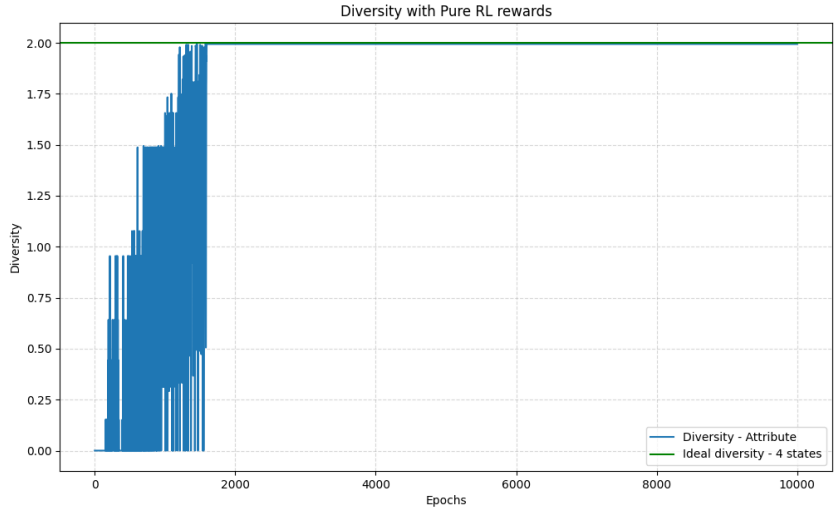


FIGURE 5.15. Diversity of pure RL over 10000 epochs for 4 states and $n = 100$ nodes. The switch frequency parameter is 0.5 and the switch cooldown is between 1 and 10.

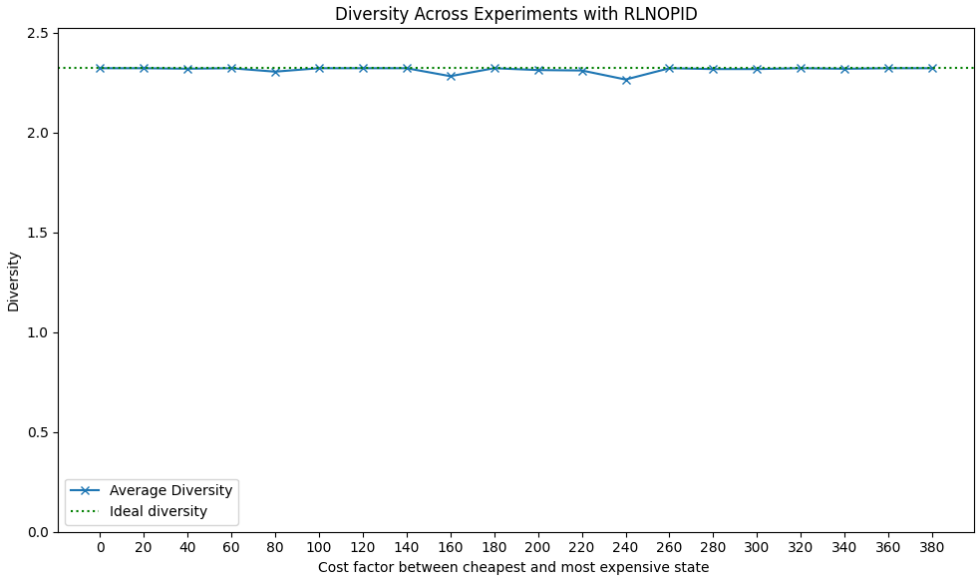


FIGURE 5.16. Diversity using pure RL at the end of 1000 epochs with a growing factor in running costs between the cheapest and most expensive state..

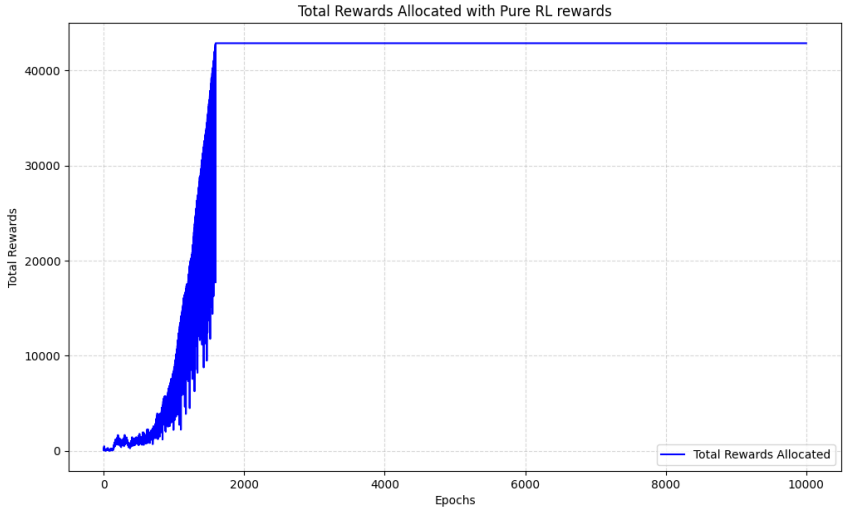


FIGURE 5.17. Total rewards of pure RL over 10000 epochs for 4 states and $n = 100$ nodes. The switch frequency parameter is 0.5 and the switch cooldown is between 1 and 10.

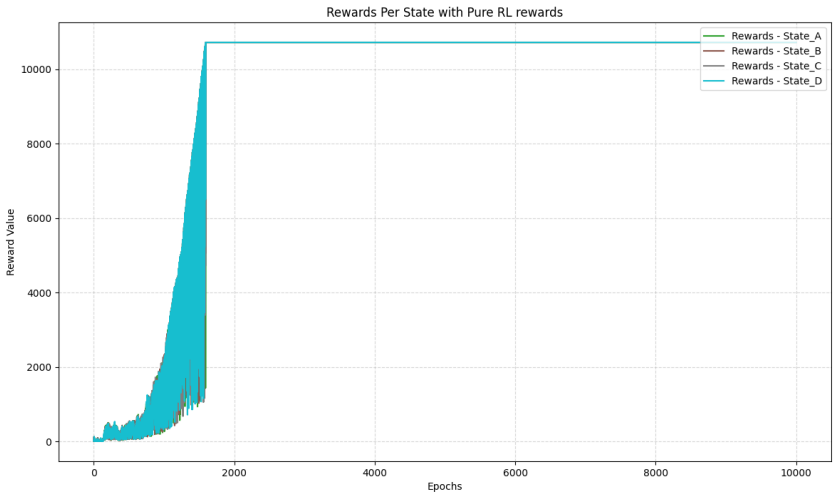


FIGURE 5.18. Rewards per state of pure RL over 10000 epochs for 4 states and $n = 100$ nodes. The switch frequency parameter is 0.5 and the switch cooldown is between 1 and 10.

5.4.4 Adding a new state while the system runs.

In a real deployment, it could be expected to add new allowed states to the system. For example, a new Ethereum execution client could be released and approved by the community.

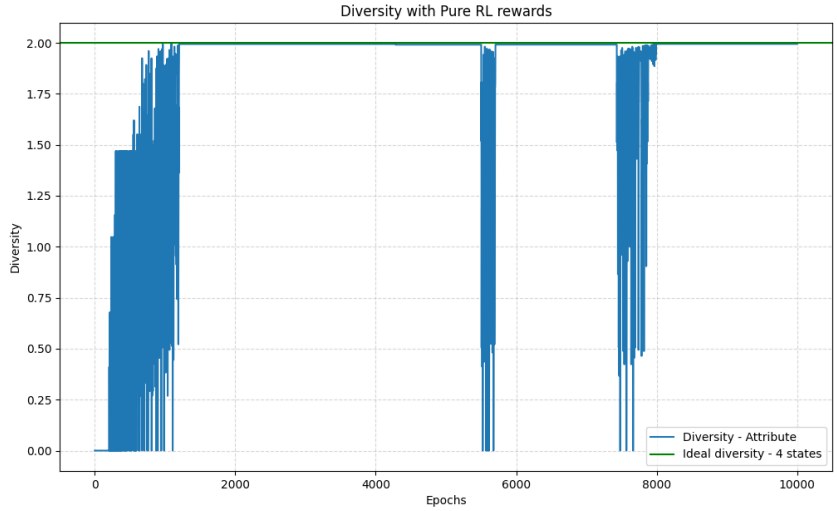


FIGURE 5.19. Diversity of pure RL over 10000 epochs for 4 states and $n = 100$ nodes with a 0.01% decrease in total rewards per epoch. The switch frequency parameter is 0.5 and the switch cooldown is between 1 and 10.

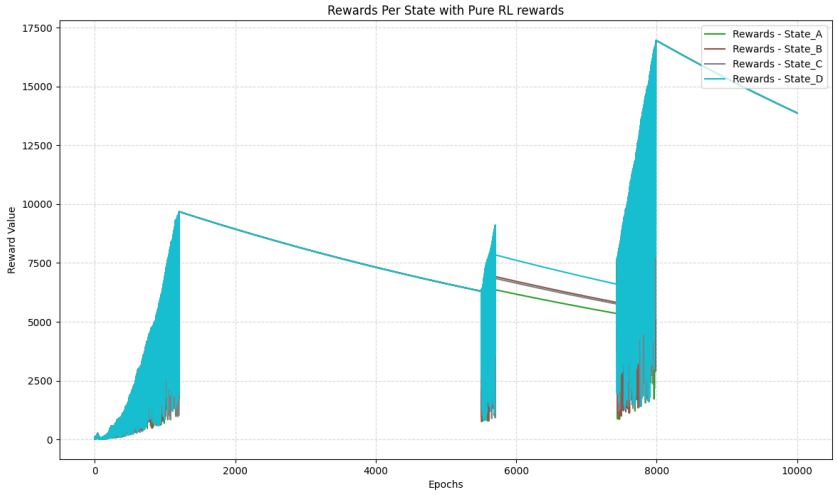


FIGURE 5.20. Rewards per state of pure RL over 10000 epochs for 4 states and $n = 100$ nodes with a 0.01% decrease in total rewards per epoch. The switch frequency parameter is 0.5 and the switch cooldown is between 1 and 10.

Our 3 methods manage to adapt and reach ideal diversity after a short period of instability, observed in Figure 5.21.

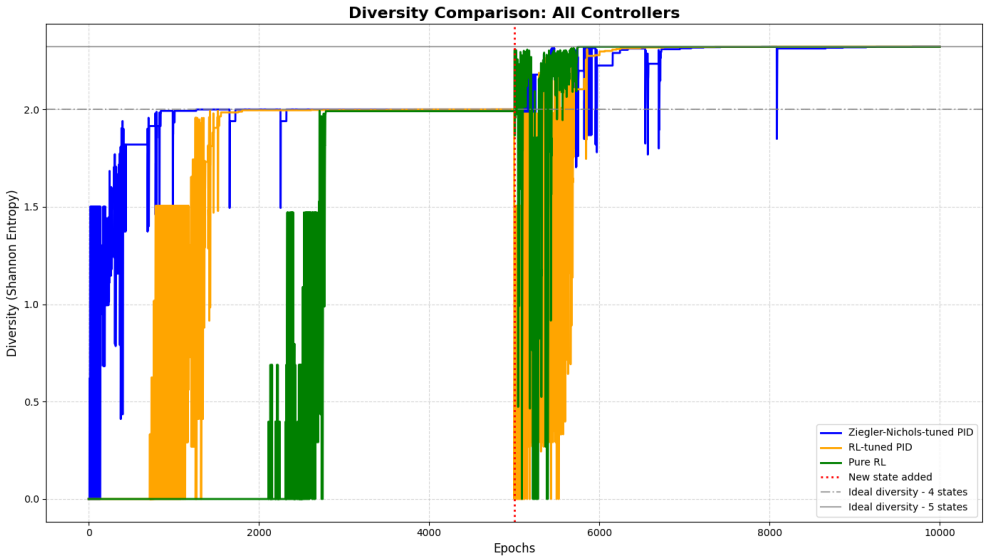


FIGURE 5.21. Diversity of the ZN-tuned PID, RL-tuned PID , and Pure RL approaches over 10000 epochs for 4 states and $n = 100$ nodes. A new state is added at epoch $e = 5000$. The switch frequency parameter is 0.5 and the switch cooldown is between 1 and 10.

5.5 Discussions and future work.

5.5.1 Discussing the methods.

How new state disrupts PID controllers. Although the RL-tuned PID and the Ziegler-Nichols-tuned PID system both adapt using a PID controller, we observe different disruptions by the new state's addition. This difference can be explained by the Ziegler-Nichols method's tendency to assign quite large values for K_P, while the RL-tuning gradually increases K_P as it learns and sets a much lower value for K_P. We compare the PID parameters of both approaches in Figure 5.22. To understand the PID behavior when a new state is added, consider that the target value suddenly changes for all states. At 4 initial states, the ideal fraction of nodes of 0.25 suddenly becomes 0.2, causing states converged at 0.25 to have a negative error of -0.5. Such a change is quickly corrected with a much larger K_P value, while the RL-tuned PID gradually adapts causing nodes to switch back and forth for more epochs.

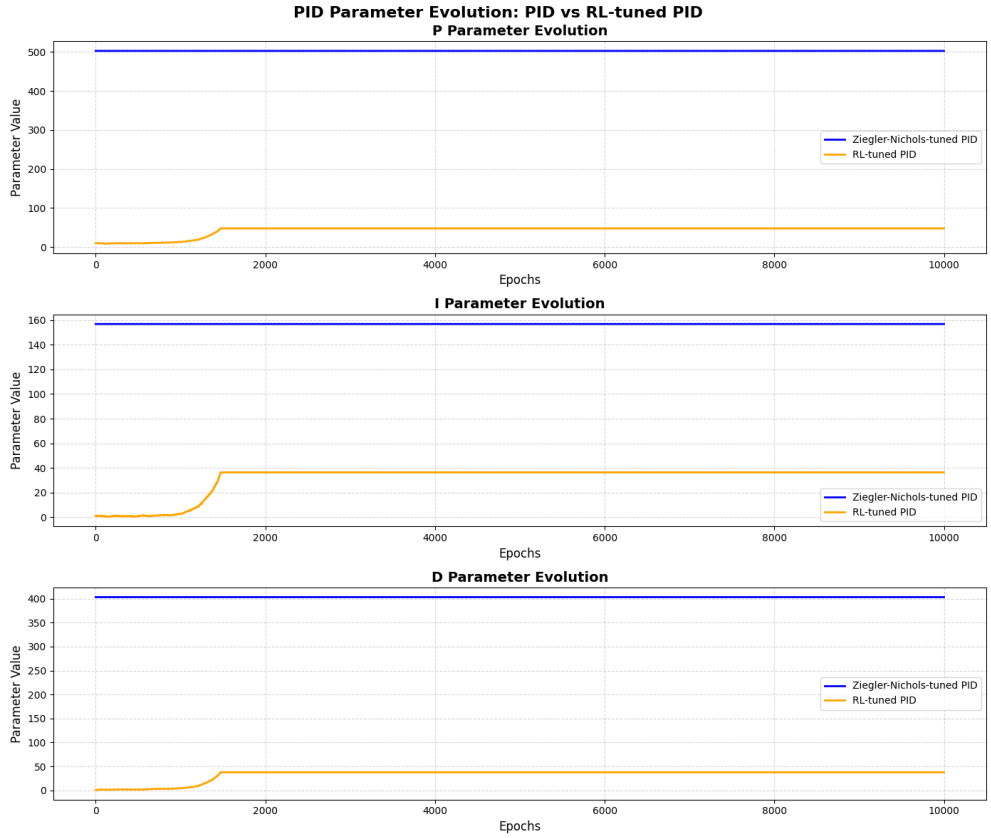


FIGURE 5.22. PID parameter evolution of the Ziegler-Nichols method (fixed) and the RL tuning over 10000 epochs for 4 states and $n = 100$ nodes. The switch frequency parameter is 0.5 and the switch cooldown is between 1 and 10.

Convergence speed Comparing the diversity of the three methods in the same condition in Figure 5.23, we highlight the faster convergence of the traditional Ziegler-Nichols method over the two RL-based methods.

Adding new state while still converging. In Figure 5.24, we can observe that if a new state is added while the system is still converging, the difference in convergence speed between all 3 approaches is the same relatively.

We can also observe that divesity initially drops after the new state is added, before converging back to the new, higher, ideal diversity.

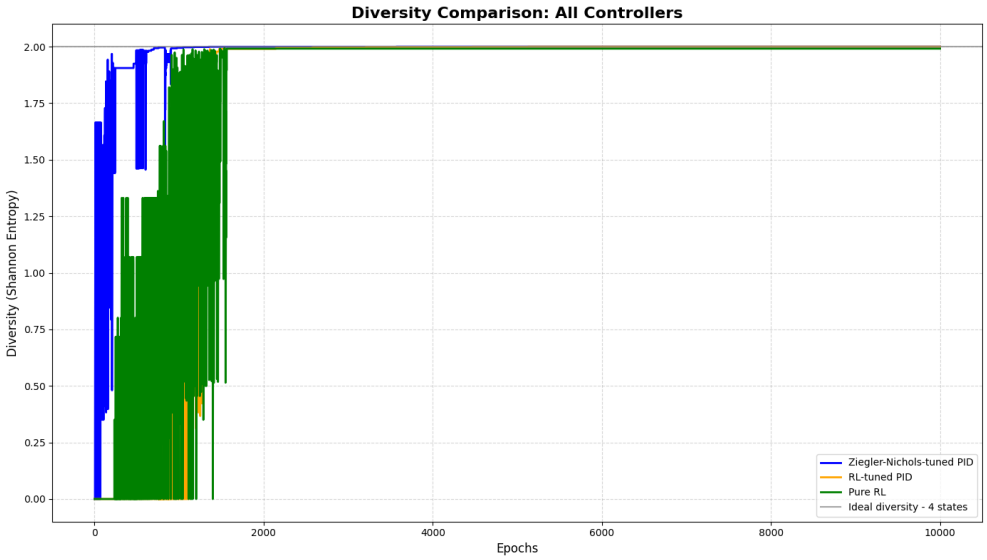


FIGURE 5.23. Diversity over 10000 epochs for 4 states and $n = 100$ nodes for the Ziegler-Nichols-tuned PID, the RL-tuned PID, and the Pure RL approach. The switch frequency parameter is 0.5 and the switch cooldown is between 1 and 10.

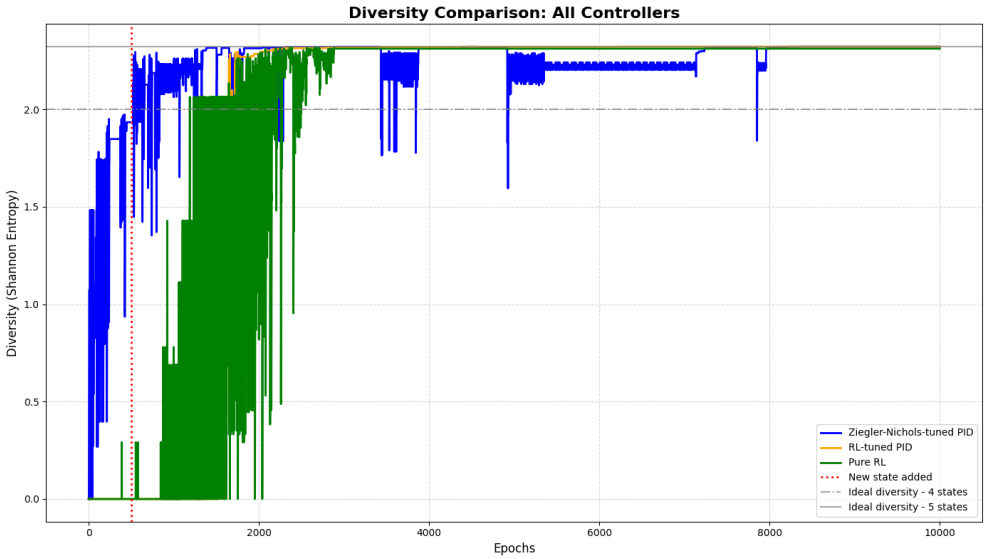


FIGURE 5.24. Diversity over 10000 epochs for 4 states and $n = 100$ nodes for the Ziegler-Nichols-tuned PID, the RL-tuned PID, and the Pure RL appraoch. A new state is added at epoch 500. The switch frequency parameter is 0.5 and the switch cooldown is between 1 and 10.

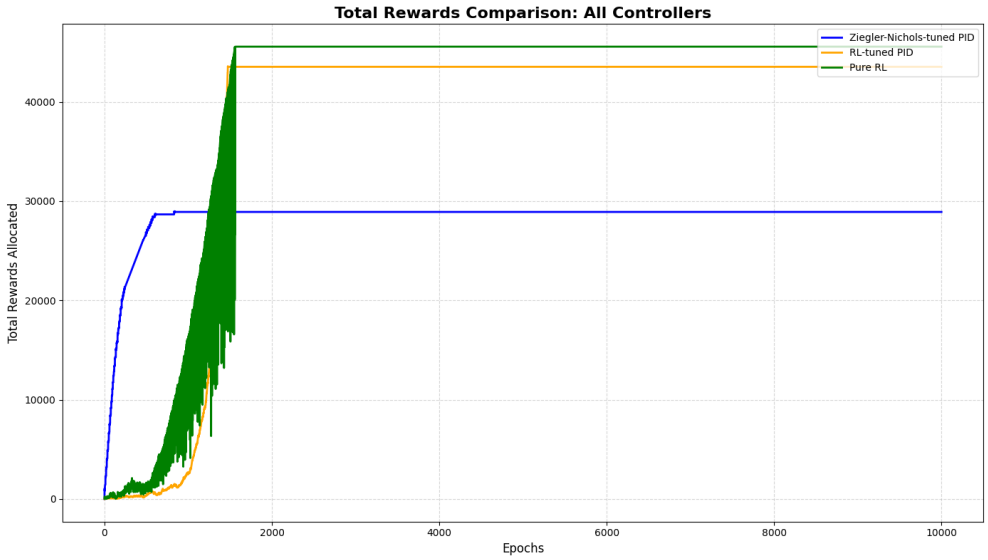


FIGURE 5.25. Total rewards comparison between Ziegler-Nichols tuned PID, RL-tuned PID, and the Pure RL approach over 10000 epochs for 4 states and $n = 100$ nodes. The switch frequency parameter is 0.5 and the switch cooldown is between 1 and 10.

Total rewards. Figure 5.25 highlights the total rewards gap caused by the pure RL approach and the PID methods.

Impact of switch frequency parameter. The switch frequency parameter determines the range of switch threshold randomly assigned to nodes. A switch frequency parameter of 0.5 as used in many of our experiments implies that some nodes may only switch if another state is 50% more profitable. All nodes' threshold is between 0% and 50%.

By setting a higher switch frequency parameter, all 3 approaches achieve better diversity and remain more stable, as shown in Figure 5.26. At low values, the results are a lot noisier, and achieved diversity is overall lower. As nodes switch less frequently, the controllers have more time to adapt and fine tune rewards for each state, increasing the stability and achieved diversity.

We observe that the RL-PID method stabilizes at a switch frequency parameter of 2.4, the Pure RL at 2.8, and the ZN-PID at 3.2. RL methods therefore may be better suited in environments where the nodes' behavior matches such switch frequency values.

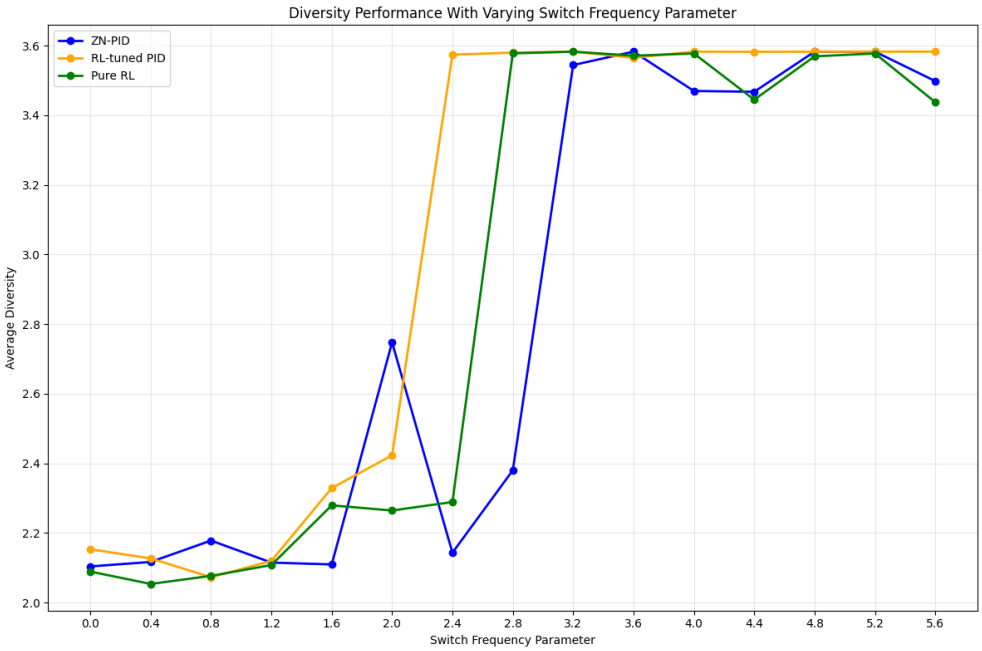


FIGURE 5.26. Average diversity in the last 25% of epochs using Ziegler-Nichols tuned PID, RL-tuned PID, and the Pure RL approach with a rising switch frequency parameter. Each experiment ran 15000 epochs for 12 states and $n = 100$ nodes. The switch cooldown is between 1 and 10.

In Figure 5.27, we observe that at a switch frequency parameter of 2, the ZN-tuned PID is able to handle the addition of a new state without any period of lower diversity, while the other methods still suffer from the unstable transition as previously observed.

Overall performance. The results on achieved diversity with rising cost gaps between states in Figures 5.2, 5.11, 5.16 show that the RL approaches are only a benefit if the real cost gaps between states are much larger (over 20x) than the Ziegler-Nichols tuning was intended for. Otherwise the traditional Ziegler-Nichols-tuned PID is able to converge faster, keep total rewards stable, and handle the addition of a new state more gracefully.

5.5.2 Future work.

Unstable transition problem.

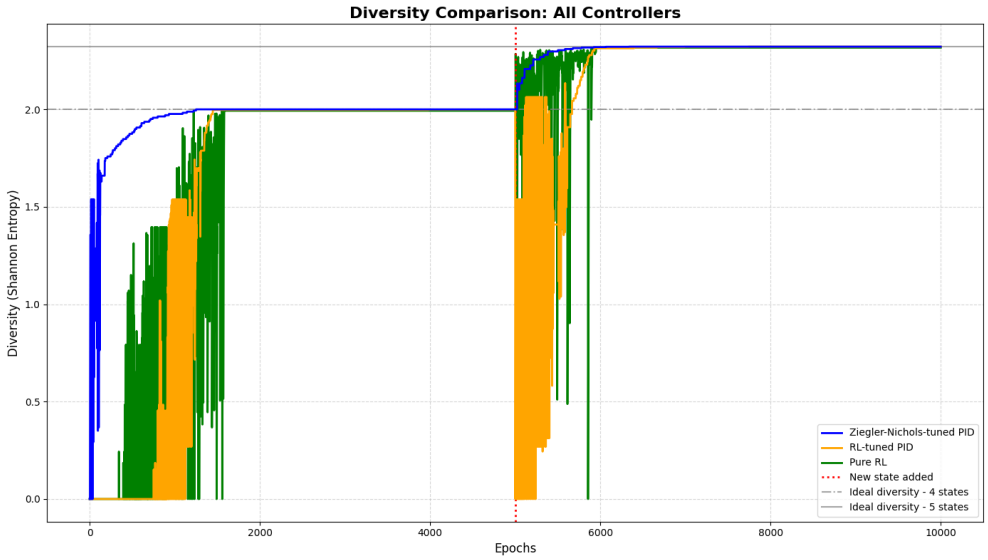


FIGURE 5.27. Diversity over 10000 epochs for 4 states and $n = 100$ nodes for the Ziegler-Nichols-tuned PID, the RL-tuned PID, and the Pure RL approach. The switch frequency parameter is 2 and the switch cooldown is between 1 and 10.

When adding a new state, in each approach, the node distribution becomes briefly unstable before converging again. As a result, for certain epochs, diversity is lower than when the system was stable around less states. Ideally the addition of a new state would lead only to diversity increases, without any decreases, even if only transitory.

Monitoring and adjusting switch frequency.

The switch frequency parameter of our simulations can artificially be scaled up or down. To obtain proper information on the requirements of a practical deployment, understanding the real world switch frequency of nodes would beneficial.

If nodes switch infrequently in practice, our methods are able to achieve ideal diversity with minimal changes, as we have seen in our results. In particular, the unstable transition problem discussed above disappears for the ZN-tuned PID method at a switch frequency parameter of 2.

If nodes switch frequently, there may be a need to design additional mechanisms to discourage frequent switching. For example, a switching cost could be added for all nodes except a

randomly selected subset, thereby skewing the switch frequency distribution of the nodes to higher values.

Assuming rational nodes. The real behavior of nodes is unpredictable, especially without any real world data available. Will they act rationally? Do they themselves know their own costs? Nodes may experiment for themselves before deciding which versions they consider low cost.

In addition, nodes may not behave according to our rational behavior during the early tuning and training phases of our systems, potentially hurting the system's ability to converge.

Colluding nodes. Nodes may also try to collude and game the system, ignoring a state on purpose to temporarily boost its future rewards. There may exist collusion strategies that provide higher rewards than the individually rational strategy, leading to poor diversity metrics.

Deploying the incentive mechanisms. To deploy any of our proposed incentive mechanisms, implementation details must be further analyzed. For example, in Ethereum's case, the overhead of implementing a mechanism such as the PID controller in a smart contract must be evaluated. It is possible that allocating the rewards may be implemented within the current reward infrastructure, but it may require a separate smart contract. In addition, collecting and verifying the attestations from participating nodes involves costs that depend on the specific attestation implementation.

CHAPTER 6

Conclusion

This thesis has addressed critical vulnerabilities in Byzantine Fault-Tolerant protocols when deployed over the internet, particularly focusing on two fundamental challenges that arise when transitioning from traditional closed systems to open blockchain networks: mobile Denial-of-Service attacks and correlated failures due to lack of fault independence. Through formal modeling, novel protocol designs, and practical incentive mechanisms, this work advances the resilience of distributed consensus systems for real-world internet deployments.

Chapter 3 introduced the Mobile Crash Adaptive Byzantine (MCAB) adversary model, providing the first systematic framework to capture mobile DoS attacks in internet-deployed consensus protocols. By defining the properties of Concealment and Abundance as necessary and sufficient conditions for liveness under mobile attacks, this work established a theoretical foundation for analyzing and designing DoS-resistant protocols. The case studies demonstrated that protocols like Algorand naturally achieve stronger security guarantees than previously proven, while traditional leader-based protocols like Hotstuff require modifications to maintain liveness. This contribution bridges the gap between informal claims about DoS resistance and formal security analysis, providing protocol designers with concrete design principles for internet deployment.

Chapter 4 expanded the knowledge on DAG-based consensus protocols by introducing the first constant latency dynamically available DAG protocol and the novel Graded Common Prefix primitive. The dynamically available DAG protocol achieves 3Δ expected latency while scaling throughput with the number of nodes, addressing a significant limitation in existing sleepy model BFT protocols. The Graded Common Prefix primitive provides an efficient alternative to full consensus for DAG finalization, requiring only 2 communication

steps compared to 4 steps for traditional BFT protocols. By synthesizing these contributions, a flexible protocol that allows clients to choose between prioritizing liveness or safety is obtained. In addition, using the MCAB model and our protocols, the Global Awake Time assumption for flexible protocols is able to be relaxed to Quorum Awake Time.

Chapter 5 tackled the critical problem of fault independence in open blockchain networks through the lens of incentive mechanism design. By formally defining single points of failure and proving that at least $\lceil n/f \rceil$ evenly distributed configuration versions are necessary to avoid such failures, this work established clear diversity requirements for resilient systems. The three proposed and evaluated adaptive reward allocation mechanisms—Ziegler-Nichols tuned PID controller, reinforcement learning tuned PID controller, and pure reinforcement learning approach—demonstrate that traditional control theory methods can effectively address the unknown cost problem in configuration diversity. Surprisingly, the simulations revealed that the classical PID controller often outperforms more sophisticated RL-based approaches, converging faster and handling dynamic changes more gracefully, while RL methods show advantages only in extreme cost variation scenarios.

Together, these contributions provide a comprehensive framework for building more resilient distributed ledgers deployed on the internet. The MCAB model offers protocol designers a systematic way to reason about and defend against mobile DoS attacks, while the DAG-based protocols demonstrate how to achieve both high performance and flexible security guarantees. The incentive mechanisms show how to address correlated failures through economic incentives rather than protocol modifications, making them deployable on existing blockchain systems. As distributed ledgers continue to scale and operate in increasingly adversarial internet environments, the theoretical foundations, efficient protocols, and incentive mechanisms developed in this thesis will be essential for maintaining security, performance, and resilience in next-generation distributed systems.

References

[1] Monero means money: Private, decentralized cryptocurrency. https://www.getmonero.org/resources/about/. Accessed: 2025-06-19.

[2] Use cases of hyperledger fabric. https://hyperledger-fabric.readthedocs.io/en/latest/usecases.html. Accessed: 2025-06-19.

[3] Welcome to ethereum: The leading platform for innovative apps and blockchain networks. https://ethereum.org/en/. Accessed: 2025-06-19.

[4] Ethereum client distribution. https://clientdiversity.org/#distribution, 2025. Accessed: 2025-06-03.

[5] Ethereum client switcher. https://github.com/accidental-green/client-switcher, 2025. Accessed: 2025-06-03.

[6] Ittai Abraham, T-H. Hubert Chan, Danny Dolev, Kartik Nayak, Rafael Pass, Ling Ren, and Elaine Shi. Communication complexity of byzantine agreement, revisited. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, PODC '19, page 317–326, New York, NY, USA, 2019. Association for Computing Machinery.

[7] Ittai Abraham, Srinivas Devadas, Danny Dolev, Kartik Nayak, and Ling Ren. Synchronous byzantine agreement with expected O(1) rounds, expected o(n^2) communication, and optimal resilience. In Ian Goldberg and Tyler Moore, editors, *Financial Cryptography and Data Security - 23rd International Conference, FC 2019, Frigate Bay, St. Kitts and Nevis, February 18-22, 2019, Revised Selected Papers*, volume 11598 of *Lecture Notes in Computer Science*, pages 320–334. Springer, 2019.

[8] Ittai Abraham, Danny Dolev, Alon Kagan, and Gilad Stern. Brief announcement: Authenticated consensus in synchronous systems with mixed faults. In *36th International Symposium on Distributed Computing (DISC)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022.

[9] Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Maofan Yin. Sync hotstuff: Simple and practical synchronous state machine replication. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 106–118. IEEE, 2020.

[10] Ittai Abraham, Dahlia Malkhi, and Alexander Spiegelman. Asymptotically optimal validated asynchronous byzantine agreement. In Peter Robinson and Faith Ellen, editors, *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*, pages 337–346. ACM, 2019.

[11] Marcos Kawazoe Aguilera and Sam Toueg. A simple bivalency proof that *t*-resilient consensus requires *t* + 1 rounds. *Inf. Process. Lett.*, 71(3-4):155–158, 1999.

[12] Marcos Kawazoe Aguilera and Sam Toueg. A simple bivalency proof that *t*-resilient consensus requires *t* + 1 rounds. *Inf. Process. Lett.*, 71(3-4):155–158, 1999.

[13] Orestis Alpos, Christian Cachin, and Luca Zanolini. How to trust strangers: Composition of Byzantine quorum systems. In *2021 40th International Symposium on Reliable Distributed Systems (SRDS)*, pages 120–131. IEEE, 2021.

[14] Kiam Heong Ang, Gregory Chong, and Yun Li. Pid control system analysis, design, and technology. *IEEE Transactions on Control Systems Technology*, 13(4):559–576, 2005.

[15] Karolos Antoniadis, Antoine Desjardins, Vincent Gramoli, Rachid Guerraoui, and Igor Zablotchi. Leaderless consensus. In *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*, pages 392–402. IEEE, 2021.

[16] Robert B Ash. *Information theory*. Courier Corporation, 2012.

[17] Karl Johan Åström and Tore Hägglund. Revisiting the ziegler–nichols step response method for pid control. *Journal of process control*, 14(6):635–650, 2004.

[18] Algirdas Avizienis. The n-version approach to fault-tolerant software. *IEEE Transactions on Software Engineering*, (12):1491–1501, 1985.

[19] Kushal Babel, Andrey Chursin, George Danezis, Anastasios Kichidis, Lefteris Kokoris-Kogias, Arun Koshy, Alberto Sonnino, and Mingwei Tian. Mysticeti: Reaching the latency limits with uncertified dags. In *Network and Distributed Systems Security Symposium (NDSS)*, 2025.

[20] Christian Badertscher, Peter Gaži, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 913–930, 2018.

[21] Christian Badertscher, Peter Gazi, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. Ouroboros chronos: Permissionless clock synchronization via proof-of-stake. *Cryptology ePrint Archive*, 2019.

[22] Vivek Bagaria, Sreeram Kannan, David Tse, Giulia Fanti, and Pramod Viswanath. Prism: Deconstructing the blockchain to approach physical limits. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 585–602, 2019.

[23] Nazreen Banu, Samia Souissi, Taisuke Izumi, and Koichi Wada. An improved Byzantine agreement algorithm for synchronous systems with mobile faults. *International Journal of Computer Applications*, 43(22):1–7, 2012.

[24] Fabrice Benhamouda, Craig Gentry, Sergey Gorbunov, Shai Halevi, Hugo Krawczyk, Chengyu Lin, Tal Rabin, and Leonid Reyzin. Can a public blockchain keep a secret? In *Theory of Cryptography Conference*, pages 260–290. Springer, 2020.

[25] Adithya Bhat, Nibesh Shrestha, Zhongtang Luo, Aniket Kate, and Kartik Nayak. Randpiper–reconfiguration-friendly random beacons with quadratic communication. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 3502–3524, 2021.

[26] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In *Annual international cryptology conference*, pages 757–788. Springer, 2018.

[27] Dan Boneh, Saba Eskandarian, Lucjan Hanzlik, and Nicola Greco. Single secret leader election. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, pages 12–24, 2020.

[28] François Bonnet, Xavier Défago, Thanh Dang Nguyen, and Maria Potop-Butucaru. Tight bound on mobile Byzantine agreement. In *International Symposium on Distributed Computing*, pages 76–90. Springer, 2014.

[29] Ethan Buchman. *Tendermint: Byzantine fault tolerance in the age of blockchains*. PhD thesis, University of Guelph, 2016.

[30] Harry Buhrman, Juan A Garay, and J-H Hoepman. Optimal resiliency against mobile faults. In *Twenty-Fifth International Symposium on Fault-Tolerant Computing. Digest of Papers*, pages 83–88. IEEE, 1995.

[31] Benedikt Bünz, Lucianna Kiffer, Loi Luu, and Mahdi Zamani. Flyclient: Super-light clients for cryptocurrencies. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 928–946. IEEE, 2020.

[32] Vitalik Buterin. The meaning of decentralization. https://medium.com/@VitalikButerin/the-meaning-of-decentralization-a0c92b76a274/, 2017. Accessed: 2025-06-03.

[33] Vitalik Buterin, Diego Hernandez, Thor Kamphefner, Khiem Pham, Zhi Qiao, Danny Ryan, Juhyeok Sin, Ying Wang, and Yan X Zhang. Combining ghost and casper. *arXiv*

preprint arXiv:2003.03052, 2020.

[34] Christian Cachin. Distributing trust on the internet. In *2001 International Conference on Dependable Systems and Networks*, pages 183–192. IEEE, 2001.

[35] Christian Cachin, Klaus Kursawe, Anna Lysyanskaya, and Reto Strobl. Asynchronous verifiable secret sharing and proactive cryptosystems. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 88–97, 2002.

[36] Miguel Castro and Barbara Liskov. Practical Byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems (TOCS)*, 20(4):398–461, 2002.

[37] Dario Catalano, Dario Fiore, and Emanuele Giunta. Adaptively secure single secret leader election from ddh. In *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 430–439, 2022.

[38] Pyrros Chaidos and Aggelos Kiayias. Mithril: Stake-based threshold multisignatures. *Cryptology ePrint Archive*, 2021.

[39] Jing Chen and Silvio Micali. Algorand: A secure and efficient distributed ledger. *Theor. Comput. Sci.*, 777:155–183, 2019.

[40] Liming Chen and Algirdas Avizienis. The methodology of n-version programming. *Software Fault Tolerance*, pages 95–113, 1995.

[41] Yu-hu CHENG, SUN Wei, et al. A proposal of adaptive pid controller based on reinforcement learning. *Journal of China University of Mining and Technology*, 17(1):40–44, 2007.

[42] George Coker, Joshua Guttman, Peter Loscocco, Amy Herzog, Jonathan Millen, Brian O'Hanlon, John Ramsdell, Ariel Segall, Justin Sheehy, and Brian Sniffen. Principles of remote attestation. *International journal of information security*, 10:63–81, 2011.

[43] P Cominos and N Munro. Pid controllers: recent tuning methods and design to specification. *IEE Proceedings-Control Theory and Applications*, 149(1):46–53, 2002.

[44] Phil Daian, Rafael Pass, and Elaine Shi. Snow white: Robustly reconfigurable consensus and applications to provably secure proof of stake. In *Financial Cryptography and Data Security: 23rd International Conference, (FC), Frigate Bay, St. Kitts and Nevis, February 18–22, 2019, Revised Selected Papers 23*, pages 23–41. Springer, 2019.

[45] Francesco D'Amato and Luca Zanolini. Streamlining sleepy consensus: Total-order broadcast with single-vote decisions in the sleepy model. *arXiv e-prints*, pages arXiv–2310, 2023.

[46] George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. Narwhal and tusk: a dag-based mempool and efficient BFT consensus. In Yérom-David Bromberg, Anne-Marie Kermarrec, and Christos Kozyrakis, editors, *EuroSys '22: Seventeenth European Conference on Computer Systems, Rennes, France, April 5 - 8, 2022*, pages 34–50. ACM, 2022.

[47] Sourav Das, Vinith Krishnan, Irene Miriam Isaac, and Ling Ren. Spurt: Scalable distributed randomness beacon with transparent setup. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 2502–2517. IEEE, 2022.

[48] Bernardo David, Peter Gaži, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 66–98. Springer, 2018.

[49] Jérémie Decouchant, David Kozhaya, Vincent Rahli, and Jiangshan Yu. DAMYSUS: streamlined BFT consensus leveraging trusted components. In Yérom-David Bromberg, Anne-Marie Kermarrec, and Christos Kozyrakis, editors, *EuroSys*, pages 1–16. ACM, 2022.

[50] Amir Dembo, Sreeram Kannan, Ertem Nusret Tas, David Tse, Pramod Viswanath, Xuechao Wang, and Ofer Zeitouni. Everything is a race and Nakamoto always wins. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 859–878, 2020.

[51] Yevgeniy Dodis, Matt Franklin, Jonathan Katz, Atsuko Miyaji, and Moti Yung. Intrusion-resilient public-key encryption. In *Cryptographers' Track at the RSA Conference*, pages 19–32. Springer, 2003.

[52] Oguzhan Dogru, Kirubakaran Velswamy, Fadi Ibrahim, Yuqi Wu, Arun Senthil Sundaramoorthy, Biao Huang, Shu Xu, Mark Nixon, and Noel Bell. Reinforcement learning approach to autonomous pid tuning. *Computers & Chemical Engineering*, 161:107760, 2022.

[53] Danny Dolev and Rüdiger Reischuk. Bounds on information exchange for Byzantine agreement. *J. ACM*, 32(1):191–204, 1985.

[54] Sisi Duan and Haibin Zhang. Foundations of dynamic bft. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 1546–1546. IEEE Computer Society, 2022.

[55] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM (JACM)*, 35(2):288–323, 1988.

[56] Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985.

[57] Sui Foundation. Sui mainnet outage resolution. https://blog.sui.io/sui-mainnet-outage-resolution/, 2024. Accessed: 2025-06-03.

[58] Aurélien Francillon, Quan Nguyen, Kasper B Rasmussen, and Gene Tsudik. A minimalist approach to remote attestation. In *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6. IEEE, 2014.

[59] Gene F Franklin, J David Powell, and Abbas Emami-Naeini. *Feedback control of dynamic systems*. Pearson, 7 edition, 2014.

[60] Yingzi Gao, Yuan Lu, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. Dumbo-ng: Fast asynchronous BFT consensus with throughput-oblivious latency. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, pages 1187–1201. ACM, 2022.

[61] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 281–310. Springer, 2015.

[62] Juan Garay, Aggelos Kiayias, and Yu Shen. Proof-of-work-based consensus in expected-constant time. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 96–125. Springer, 2024.

[63] Juan A Garay. Reaching (and maintaining) agreement in the presence of mobile faults. In *International Workshop on Distributed Algorithms*, pages 253–264. Springer, 1994.

[64] Miguel Garcia, Alysson Bessani, and Nuno Neves. Lazarus: Automatic management of diversity in bft systems. In *Proceedings of the 20th International Middleware Conference*, pages 241–254, 2019.

[65] Rati Gelashvili, Lefteris Kokoris-Kogias, Alberto Sonnino, Alexander Spiegelman, and Zhuolun Xiang. Jolteon and ditto: Network-adaptive efficient consensus with asynchronous fallback. In Ittay Eyal and Juan A. Garay, editors, *Financial Cryptography and Data Security - 26th International Conference, FC 2022, Grenada, May 2-6, 2022, Revised Selected Papers*, volume 13411 of *Lecture Notes in Computer Science*, pages 296–315. Springer, 2022.

[66] Craig Gentry, Shai Halevi, Hugo Krawczyk, Bernardo Magri, Jesper Buus Nielsen, Tal Rabin, and Sophia Yakoubov. Yoso: you only speak once. In *Annual International Cryptology Conference*, pages 64–93. Springer, 2021.

[67] 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*, pages 51–68, 2017.

[68] Seth Gilbert and Nancy A. Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33(2):51–59, 2002.

[69] Neil Giridharan, Florian Suri-Payer, Matthew Ding, Heidi Howard, Ittai Abraham, and Natacha Crooks. Beegees: Stayin' alive in chained bft. In *Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing*, PODC '23, page 233–243, New York, NY, USA, 2023. Association for Computing Machinery.

[70] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[71] Vipul Goyal, Abhiram Kothapalli, Elisaweta Masserova, Bryan Parno, and Yifan Song. Storing and retrieving secrets on a blockchain. In *IACR International Conference on Public-Key Cryptography*, pages 252–282. Springer, 2022.

[72] Vipul Goyal, Hanjun Li, and Justin Raizes. Instant block confirmation in the sleepy model. In *International Conference on Financial Cryptography and Data Security (FC)*, pages 65–83. Springer, 2021.

[73] Guy Golan Gueta, Ittai Abraham, Shelly Grossman, Dahlia Malkhi, Benny Pinkas, Michael Reiter, Dragos-Adrian Seredinschi, Orr Tamir, and Alin Tomescu. Sbft: a scalable and decentralized trust infrastructure. In *2019 49th Annual IEEE/IFIP international conference on dependable systems and networks (DSN)*, pages 568–580. IEEE, 2019.

[74] Bingyong Guo, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. Dumbo: Faster asynchronous BFT protocols. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, pages 803–818. ACM, 2020.

[75] Yue Guo, Rafael Pass, and Elaine Shi. Synchronous, with a chance of partition tolerance. In *Annual International Cryptology Conference*, pages 499–529. Springer, 2019.

[76] Alyssa Hertig. So, ethereum's blockchain is still under attack.... https://www.coindesk.com/markets/2016/10/06/so-ethereums-blockchain-is-still-under-attack/, 2016. Accessed: 2023-04-06.

[77] Amir Herzberg, Stanisław Jarecki, Hugo Krawczyk, and Moti Yung. Proactive secret sharing or: How to cope with perpetual leakage. In *annual international cryptology conference*, pages 339–352. Springer, 1995.

[78] Kurt Hornik, Maxwell B. Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[79] IOHK. Cardano docs: Ouroboros overview. https://docs.cardano.org/learn/ouroboros-overview/, 2023. Accessed: 2023-12-28.

[80] Mohammad M Jalalzai, Jianyu Niu, Chen Feng, and Fangyu Gai. Fast-hotstuff: A fast and resilient hotstuff protocol. *arXiv preprint arXiv:2010.11454*, 2020.

[81] Flavio Paiva Junqueira, Ranjita Bhagwan, Alejandro Hevia, Keith Marzullo, and Geoffrey M Voelker. Surviving internet catastrophes. In *USENIX Annual Technical Conference, General Track*, pages 45–60, 2005.

[82] Aggelos Kiayias, Andrew Miller, and Dionysis Zindros. Non-interactive proofs of proof-of-work. In *International Conference on Financial Cryptography and Data Security*, pages 505–522. Springer, 2020.

[83] Justin Kim, Vandan Mehta, Kartik Nayak, and Nibesh Shrestha. Brief announcement: Making synchronous BFT protocols secure in the presence of mobile sluggish faults. In *ACM Symposium on Principles of Distributed Computing (PODC), Virtual Event, Italy, July 26-30, 2021*, pages 375–377. ACM, 2021.

[84] Klaus Kursawe and Felix C Freiling. *Byzantine fault tolerance on general hybrid adversary structures*. RWTH, Department of Computer Science, 2005.

[85] Andrew Lewis-Pye, Kartik Nayak, and Nibesh Shrestha. The pipes model for latency analysis. *Cryptology ePrint Archive*, 2025.

[86] Chao Liu, Sisi Duan, and Haibin Zhang. Epic: efficient asynchronous bft with adaptive security. In *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 437–451. IEEE, 2020.

[87] Lostin. A complete history of solana outages: Causes, fixes, and lessons learnt. https://www.helius.dev/blog/solana-outages-complete-history/, 2025. Accessed: 2025-06-03.

[88] Yuan Lu, Zhenliang Lu, and Qiang Tang. Bolt-dumbo transformer: Asynchronous consensus as fast as the pipelined BFT. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, pages 2159–2173. ACM, 2022.

[89] Yuan Lu, Zhenliang Lu, Qiang Tang, and Guiling Wang. Dumbo-mvba: Optimal multi-valued validated asynchronous Byzantine agreement, revisited. In *Proceedings of the 39th Symposium on Principles of Distributed Computing (PODC)*, pages 129–138, 2020.

[90] Hassan Maishera. Solana's latest ddos attack leads to poor network performance. https://finance.yahoo.com/news/

solana-latest-ddos-attack-leads-120022342.html, 2022. Accessed: 2023-04-06.

[91] Dahlia Malkhi, Atsuki Momose, and Ling Ren. Towards practical sleepy bft. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 490–503, 2023.

[92] Dahlia Malkhi and Kartik Nayak. Extended abstract: Hotstuff-2: Optimal two-phase responsive BFT. *IACR Cryptol. ePrint Arch.*, page 397, 2023.

[93] Dahlia Malkhi, Kartik Nayak, and Ling Ren. Flexible byzantine fault tolerance. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, page 1041–1053, New York, NY, USA, 2019. Association for Computing Machinery.

[94] Dahlia Malkhi, Chrysoula Stathakopoulou, and Maofan Yin. Bbca-chain: Low latency, high throughput bft consensus on a dag. In *International Conference on Financial Cryptography and Data Security*, pages 51–73. Springer, 2024.

[95] Sai Krishna Deepak Maram, Fan Zhang, Lun Wang, Andrew Low, Yupeng Zhang, Ari Juels, and Dawn Song. Churp: dynamic-committee proactive secret sharing. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 2369–2386, 2019.

[96] Silvio Micali, Michael Rabin, and Salil Vadhan. Verifiable random functions. In *40th annual symposium on foundations of computer science (cat. No. 99CB37039)*, pages 120–130. IEEE, 1999.

[97] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of bft protocols. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 31–42, 2016.

[98] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[99] Atsuki Momose and Ling Ren. Optimal communication complexity of authenticated byzantine agreement. In *35th International Symposium on Distributed Computing (DISC 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

[100] Atsuki Momose and Ling Ren. Constant latency in sleepy consensus. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, pages 2295–2308. ACM, 2022.

[101] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, pages 807–814, 2010.

[102] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review*, page 21260, 2008.

[103] John F Nash Jr. Equilibrium points in n-person games. *Proceedings of the national academy of sciences*, 36(1):48–49, 1950.

[104] Joachim Neu, Ertem Nusret Tas, and David Tse. Ebb-and-flow protocols: A resolution of the availability-finality dilemma. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 446–465. IEEE, 2021.

[105] Aidan O'Dwyer. Handbook of pi and pid controller tuning rules. *Imperial College Press*, 2003.

[106] Rafail Ostrovsky and Moti Yung. How to withstand mobile virus attacks. In *Proceedings of the tenth annual ACM symposium on Principles of distributed computing*, pages 51–59, 1991.

[107] Rafael Pass and Elaine Shi. The sleepy model of consensus. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 380–409. Springer, 2017.

[108] Matthieu Rambaud and Antoine Urban. Asynchronous dynamic proactive secret sharing under honest majority: Refreshing without a consistent view on shares. *Cryptology ePrint Archive*, 2022.

[109] Javier Ron, César Soto-Valero, Long Zhang, Benoit Baudry, and Martin Monperrus. Highly available blockchain nodes with n-version design. *IEEE Transactions on Dependable and Secure Computing*, 21(4):4084–4097, 2023.

[110] Toru Sasaki, Yukiko Yamauchi, Shuji Kijima, and Masafumi Yamashita. Mobile Byzantine agreement on arbitrary network. In *International Conference On Principles Of Distributed Systems*, pages 236–250. Springer, 2013.

[111] Hans Schmiedel, Runchao Han, Qiang Tang, Ron Steinfeld, and Jiangshan Yu. Modeling mobile crash in byzantine consensus. *IEEE 37th Computer Security Foundations Symposium*, 2024.

[112] David Schultz, Barbara Liskov, and Moses Liskov. Mpss: mobile proactive secret sharing. *ACM Transactions on Information and System Security (TISSEC)*, 13(4):1–32, 2010.

[113] Caspar Schwarz-Schilling, Joachim Neu, Barnabé Monnot, Aditya Asgaonkar, Ertem Nusret Tas, and David Tse. Three attacks on proof-of-stake ethereum. In *International Conference on Financial Cryptography and Data Security*, pages 560–576. Springer, 2022.

[114] Marco Serafini, Péter Bokor, Dan Dobre, Matthias Majuntke, and Neeraj Suri. Scrooge: Reducing the costs of fast Byzantine replication in presence of unresponsive replicas. In *2010 IEEE/IFIP International Conference on Dependable Systems & Networks (DSN)*, pages 353–362. IEEE, 2010.

[115] Arvind Seshadri, Adrian Perrig, Leendert Van Doorn, and Pradeep Khosla. Swatt: Software-based attestation for embedded devices. In *IEEE Symposium on Security and Privacy, 2004. Proceedings. 2004*, pages 272–282. IEEE, 2004.

[116] Nibesh Shrestha, Rohan Shrothrium, Aniket Kate, and Kartik Nayak. Sailfish: Towards Improving the Latency of DAG-Based BFT . In *2025 IEEE Symposium on Security and Privacy (SP)*, pages 1928–1946, Los Alamitos, CA, USA, May 2025. IEEE Computer Society.

[117] Sean W Smith. Outbound authentication for programmable secure coprocessors. In *Computer Security—ESORICS 2002: 7th European Symposium on Research in Computer Security Zurich, Switzerland, October 14–16, 2002 Proceedings 7*, pages 72–89. Springer, 2002.

[118] Yonatan Sompolinsky, Shai Wyborski, and Aviv Zohar. Phantom ghostdag: a scalable generalization of nakamoto consensus: September 2, 2021. In *Proceedings of the 3rd ACM Conference on Advances in Financial Technologies*, pages 57–70, 2021.

[119] Yonatan Sompolinsky and Aviv Zohar. Secure high-rate transaction processing in bitcoin. In *International conference on financial cryptography and data security (FC)*, pages 507–527. Springer, 2015.

[120] Paulo Sousa, Alysson Neves Bessani, Miguel Correia, Nuno Ferreira Neves, and Paulo Verissimo. Resilient intrusion tolerance through proactive and reactive recovery. In *13th Pacific Rim International Symposium on Dependable Computing (PRDC)*, pages 373–380. IEEE, 2007.

[121] Alexander Spiegelman, Neil Giridharan, Alberto Sonnino, and Lefteris Kokoris-Kogias. Bullshark: Dag bft protocols made practical. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 2705–2718, 2022.

[122] Alexander Spiegelman, Arik Rinberg, and Dahlia Malkhi. ACE: abstract consensus encapsulation for liveness boosting of state machine replication. In Quentin Bramas, Rotem Oshman, and Paolo Romano, editors, *24th International Conference on Principles of Distributed Systems, OPODIS 2020, December 14-16, 2020, Strasbourg, France (Virtual Conference)*, volume 184 of *LIPIcs*, pages 9:1–9:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[123] Chrysoula Stathakopoulou, Tudor David, Matej Pavlovic, and Marko Vukolic. [solution] mir-bft: Scalable and robust BFT for decentralized networks. *J. Syst. Res.*, 2(1), 2022.

[124] Xiao Sui, Sisi Duan, and Haibin Zhang. Marlin: Two-phase BFT with linearity. In *52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks, (DSN) 2022, Baltimore, MD, USA, June 27-30, 2022*, pages 54–66. IEEE, 2022.

[125] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, pages 2094–2100, 2016.

[126] Qing-Guo Wang, Chang-Chieh Hang, and Qiang Bi. Relay feedback auto-tuning of process controllers — a tutorial review. *Journal of Process Control*, 9(2):143–175, 1999.

[127] Qing-Guo Wang, Yong Zhang, and Min-Sen Chiu. Relay-based estimation of multiple points on process frequency response. *Automatica*, 33(9):1753–1757, 1997.

[128] Yunzhou Yan, Yu Xia, and Srinivas Devadas. Shanrang: Fully asynchronous proactive secret sharing with dynamic committees. *Cryptology ePrint Archive*, 2022.

[129] Lei Yang, Seo Jin Park, Mohammad Alizadeh, Sreeram Kannan, and David Tse. {DispersedLedger}:{High-Throughput} byzantine consensus on variable bandwidth networks. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 493–512, 2022.

[130] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan-Gueta, and Ittai Abraham. Hotstuff: BFT consensus with linearity and responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*, pages 347–356. ACM, 2019.

[131] Jiangshan Yu. Fault independence in blockchain. In *2023 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks-Supplemental Volume (DSN-S)*, pages 117–121. IEEE, 2023.

[132] Lidong Zhou, Fred B Schneider, and Robbert Van Renesse. Coca: A secure distributed online certification authority. *ACM Transactions on Computer Systems (TOCS)*, 20(4):329–368, 2002.

[133] Lidong Zhou, Fred B Schneider, and Robbert Van Renesse. Apss: Proactive secret sharing in asynchronous systems. *ACM transactions on information and system security (TISSEC)*, 8(3):259–286, 2005.

[134] John G Ziegler and Nathaniel B Nichols. Optimum settings for automatic controllers. *Transactions of the American Society of Mechanical Engineers*, 64(8):759–765, 1942.