Aegis: A Decentralized Expansion Blockchain

Yogev Bar-On[‡∞], Roi Bar-Zur[⋆∞], Omer Ben-Porat[⋆],
Nimrod Cohen[∞], Ittay Eyal[⋆], Matan Sitbon[∞]

[‡]Tel Aviv University [∞]Eoracle [⋆]Technion

Abstract

Blockchains implement monetary systems operated by *committees* of *nodes*. The robustness of established blockchains presents an opportunity to leverage their infrastructure for creating *expansion chains*. Expansion chains can enhance scalability, provide additional functionality to the *primary* chain they leverage, or implement separate functionalities, while benefiting from the primary chain's security and the stability of its tokens. Indeed, tools like Ethereum's EigenLayer enable nodes to stake (deposit collateral) on a primary chain to form a committee responsible for operating an expansion chain.

But here is the rub. Classical protocols assume correct, well-behaved nodes stay correct indefinitely. Yet in our case, the stake incentivizes correctness—it will be slashed (revoked) if its owner deviates. Once a node withdraws its stake, there is no basis to assume its correctness.

To address the new challenge, we present *Aegis*, an expansion chain based on primary-chain stake, assuming a bounded primary-chain write time. Aegis uses references from Aegis blocks to primary blocks to define committees, checkpoints on the primary chain to perpetuate decisions, and resets on the primary chain to establish a new committee if the previous one becomes obsolete. It ensures safety at all times and rapid progress when latency among Aegis nodes is low.

1 Introduction

Blockchains like Ethereum [63] implement monetary systems and allow users to deploy *smart contracts* [58] with arbitrary logic. The users issue *transactions* to exchange assets and interact with smart contracts. Blockchain *nodes* aggregate transactions into blocks, forming a chain that determines the system's state. However, as blockchain adoption grows, scalability has emerged as a critical challenge [19, 65]. Blockchains have limited throughput, often high latency, and are limited to deterministic execution.

To address these scalability issues and implement new functionality, numerous systems expand a *primary blockchain* in various ways [37]. Some systems improve scalability by offloading computation, thereby enhancing performance and reducing costs [56, 31]. Other expansion systems serve complementary purposes. These include oracles that aggregate external data for smart contract interactions [18, 26], and mechanisms for generating verifiable randomness [17].

In some cases, it is possible to implement an expansion as a separate *Proof of Stake (PoS)* blockchain. PoS protocols allow users to *stake*—lock their tokens as collateral [34, 21, 22]. Until they *unstake*—withdraw the collateral—they are *active* nodes, allowed to create new blocks. If they misbehave, they are penalized via *slashing* [14], losing all or part of their collateral. This approach works well for major blockchains that have reached a critical mass of participants and assets [1]. However, smaller nascent systems do not have sufficient reputation to rely on long-term participation of established nodes. Such nodes help new nodes that join the system to overcome

long-range attacks [13, 23], where an adversary presents an alternative history, indistinguishable from the true one. Additionally, in smaller systems, the value of tokens and hence the collateral amounts are volatile, leaving the system vulnerable to hijacking by a well-funded adversary.

To overcome this, stake can be managed on the primary blockchain, taking advantage of its decentralization and reliability: The nodes maintaining the expansion blockchain first lock their stake on the primary chain. Then, they agree on the series of blocks using a consensus protocol [16]. Managing stake for external tasks is already possible on major blockchains. EigenLayer [60] on Ethereum and Subnets [38] on Avalanche [52] are prominent examples. If nodes misbehave, they are penalized by losing their deposit on the primary chain. The underlying consensus protocol of the primary chain guarantees safety and liveness assuming that a sufficient majority of nodes are *correct*, i.e., follow the protocol.

Prior work (§2) extensively explored *reconfiguration* of nodes in Byzantine Fault Tolerant (BFT) systems [44]. Expansion chains like Polygon [42] and Avalanche subnets [38] have adopted these approaches. However, these systems rely on strong assumptions, such as synchronous communication among nodes or that nodes remain correct indefinitely. In practice, these assumptions may not hold: A user can unstake on the primary chain at any time, removing their collateral and hence their incentive to follow the expansion chain protocol. Thus, a node that is active when it is defined in a primary block might not be active by the time an expansion chain block is generated. Additionally, like the aforementioned PoS protocols, such expansion chains remain vulnerable to long-range attacks.

To address these challenges, particularly the vulnerability to long-range attacks and the potential for nodes to become Byzantine after unstaking, we first outline a model (§3). Nodes stake and unstake using the primary chain. Unstaking does not immediately withdraw the stake, but only after a delay Δ_{active}. A set of nodes that have stake locked at the same time are called a *committee*. Until any of them completed unstaking, the committee is *active*. As is standard [34, 50], we assume that at all times a sufficient majority of the nodes in each active committee are correct. The model employs a hybrid communication approach: nodes communicate asynchronously with each other until an unknown time t_{GST}, after which message propagation is bounded, as in the partial synchrony model [25]. Nevertheless, at all times, nodes have synchronous access to read the current state of the primary chain (as when running an Ethereum node) and can write to it in bounded time (*cf.* [33]). This combination leverages the reliability of established blockchains while accounting for potential delays in internode communication in new protocols.

We present Aegis[1] (§4), an expansion blockchain that derives its security from a primary chain. It could be used to provide additional functionality to a primary chain, or more generally, for a blockchain unrelated to the primary chain, other than using its tokens as collateral.

The protocol includes a primary-chain contract that allows nodes to checkpoint the Aegis state. The nodes construct a blockchain in which each *Aegis block* references both its predecessor and a corresponding *primary block* (Figure 1). The committee defined by that primary block is the committee in charge of generating the next Aegis block using a consensus protocol. While maintaining the Aegis chain, its nodes occasionally checkpoint the latest block with the primary-chain contract. In the happy flow, the checkpoints are less than Δ_{active} apart, so there is always an active committee.

However, creating an Aegis block might take too long due to internode communication delays. If it takes longer than Δ_{active}, there would be nothing to checkpoint. In this case, any Aegis node can issue a *reset* on the primary chain to specify a new committee. At this point, the committee is

[1]Named after the powerful divine shield carried by Zeus and Athena in Greek mythology, which first appears in Homer's Iliad.

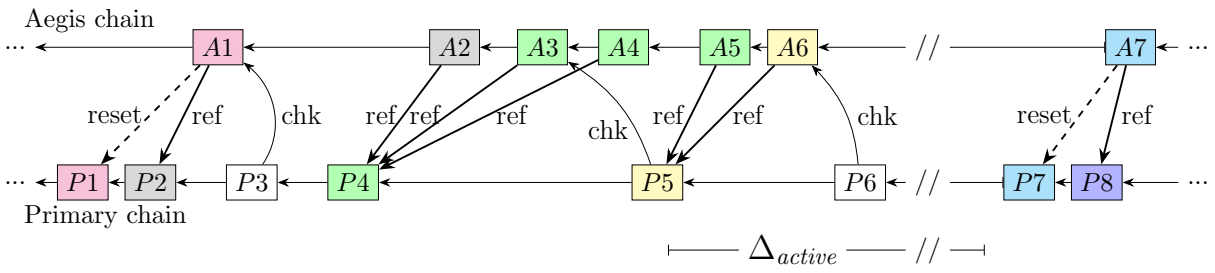

Figure 1: Aegis and primary-chain blocks.

defined in the primary-chain reset, but not yet in any Aegis block, until one is generated. The risk is that the committee defined by the reset and the one defined by the previous block both generate a block at the same height (distance from chain root), forming a so-called *fork*. To prevent that, both the protocol and the primary-chain contract enforce timing constraints, avoiding consensus decisions that cannot be checkpointed and premature resets.

To prove the security of the protocol (§5) we show that correct Aegis nodes never disagree on the blockchain content at any height. First, we use backward induction to show that if a block is created by an active committee, then all of its ancestors were created by active committees. This holds because this is verified for every block either by its child's committee or by the primary blockchain in a checkpoint. Then, using forward induction, we show that if all nodes agree up to some height, a classical consensus protocol guarantees the necessary properties for the next block. Similar arguments show validity, namely that if all nodes are correct and have the same input value, no other value is logged. We then show that after t_{GST} the protocol guarantees an active committee forms and extends the blockchain. Straightforward indistinguishability arguments bridge the gap between the static committee of the standard consensus protocol and the dynamic committees of Aegis.

Aegis can be readily implemented to decentralize various services as expansion chains. We conclude (§6) with practical considerations for performance and security.

2 Related Work

To the best of our knowledge, previous work on dynamic committees and reconfiguration in BFT systems did not consider nodes that stop being correct after their tenure (§2.1). Various work on expansion chains addressed the problem under this assumption (§2.2) and overcoming long-range attacks with checkpointing (§2.3). Others utilize multiple chains to form a new one (§2.4). Aegis can be used by Layer-2 rollups that aggregate transactions (§2.5), but it solves a distinct problem.

2.1 Classical Reconfiguration

The reconfiguration problem received intensive treatment in the classical distributed-systems literature [2]. In this setting, committees are not defined by stake on the primary chain, as in our setting, but are rather defined by an external process.

Lamport et al. [43] implement reconfiguration in a crash-fault model using a separate, trusted *configuration master*. The nodes in one committee can trigger a reconfiguration by calling the configuration master, allowing the protocol to handle many failures by pruning out crashed nodes. However, this approach relies on having the new committee obtain the previous state from the previous committee, which is not reliable in a Byzantine model, particularly in our model where after deprecation the old committee behaves arbitrarily.

Aubin et al. [4] use black-box individual consensus algorithms. On reconfiguration, they require the previous committee to agree on which entries are aborted, such that this committee will not decide on them. This is also not possible in our model, where the nodes might become deprecated due to unstaking before completing the abort.

Abraham and Malkhi [2] identify that reconfiguring protocols rely on *wedging*, where each consensus result is stored by a non-byzantine replica. This replica can be, e.g., a larger set of replicas; it can be mapped to the primary chain in our model. They propose a synchronous reconfiguration protocol, but it relies on bounded latency among nodes to identify the wedging state. This is not possible in our model, where due to asynchrony previous committees might be deprecated before taking part in this protocol.

PBFT [16] and Zyzzyva [35] work in a fully asynchronous environment and use $3f + 1$ replicas for reconfiguration. However, their progress relies on access to this reconfiguration master, implying that in case of reconfiguration events, the latency is at least long enough to read and write to the primary chain, even after t_{GST}. This is not acceptable for our purposes. On the other hand, note our model is stronger: We assume synchronous access to the primary chain.

Matchmaker [62] assumes crash-fault only, and that nodes can access deprecated nodes. They explicitly assume that joining nodes access a *matchmaker* (implemented as a set of processes, mapped to our primary chain) to obtain the current state. All access is asynchronous, allowing to complete the access to the matchmakers. Mapping to our model, this would imply that the time to create every block is at least the time to read and write to the primary chain. Again, this is unacceptable for our purposes.

Kuznetsov and Tonkikh [36] consider slow clients, that might access a deprecated committee that became Byzantine and can behave arbitrarily. To overcome this, nodes are required to destroy deprecated private keys. However, this assumption does not apply to our model. We can't assume that nodes would delete their old keys, since in our model, we assume that nodes of an active committee only follow the protocol due to the threat of slashing. But, slashing for not deleting old keys is not possible, as no one can tell when a node doesn't delete its old keys.

Note that the cooldown period for unstaking, Δ_{active}, does allow bounded-time access to previous committees that are still active. This assumption is weaker than in prior work where previous committees remain active indefinitely. Nevertheless, it allows us to achieve the desired guarantees with Aegis.

2.2 Expansion Chains

Expansion chains are blockchains that rely on a primary chain for security. They can be implemented using EigenLayer [60], Avalanche subnets [38] and Polygon 2.0 [42]. Compared to a new blockchain, an expansion chain is easier to bootstrap, as it can rely on the security of the primary chain. Nodes can be required to stake on the primary chain, using its tokens as collateral to ensure their good behavior. Due to the established security of the primary chain, the value of its tokens is more stable, and the nodes are more likely to remain correct.

On the surface, securing an expansion chain seems like the classical State Machine Replication (SMR) [45] with dynamic committee *reconfiguration* [44]. Essentially, the committee for expansion block k agrees not only on the block but also on the committee for the subsequent expansion block $k+1$, as defined in a primary block. However, this approach relies on an assumption that does not hold in our case, namely, that a correct node remains correct indefinitely. But the generation of expansion block $k + 1$ could take arbitrarily long; meanwhile, the members of its committee might unstake in the primary chain, so they are no longer incentivized by collateral. The result is either a deadlock (if we ignore blocks by a deprecated committee) or a block generated by a committee

that is no longer incentivized to follow the protocol. Aegis addresses this issue by using references from the expansion blocks to the primary blocks to define the committee for the next block, and allowing for a reset if the committee becomes obsolete before it generates a block.

BMS [57] is a mechanism for secure reconfiguration of blockchains via a trusted platform that manages reconfiguration process. However, it requires expansion chains to approve each reconfiguration, which is not always possible: If a node wishes to withdraw their stake, it is not generally acceptable to wait for such an approval, particularly if the expansion chain is asynchronous and the time is unbounded. Aegis does not require such approval, and nodes can independently withdraw without affects its security.

Avalanche subnets [38] are blockchains whose validators are defined, added and removed by transactions on a primary chain. Avalanche proposes this mechanism and facilitates open participation in its so-called elastic subnets. EigenLayer [60] allows Ethereum validators to *restake* their Ethereum-protocol deposits, committing to validating a secondary system with the same stake used for Ethereum, meaning that the same deposit can be slashed due to misbehavior in either system. Both mechanisms provide a new tool for creating expansion protocols, but leave open the question of securing the protocol, which Aegis addresses as mentioned above.

Polygon 2.0 [42] secures a secondary blockchain using Ethereum as primary blockchain. It uses staking on Ethereum to define the nodes maintaining separate blockchains. However, it encounters the aforementioned issue of SMRs with dynamic reconfiguration, meaning its security implicitly relies on either synchronous communication among Polygon nodes or that correct nodes remain correct indefinitely. This is because node set changes are events on Ethereum [40] and are voted in Polygon [41]. Therefore, if it can take arbitrarily long to reach consensus and members are not indefinitely correct, by the time they reach consensus they might not be correct. In contrast, Aegis does not rely on either assumption, while always ensuring safety and ensuring progress after t_{GST}.

Merge-mining is another approach to secure expansion blockchains [11]. It is a technique for Proof-of-Work (PoW) systems [46, 55], where nodes are called *miners* and they generate blocks by guessing hash preimages. In merge-mining, a Proof of Work miner mines on two PoW blockchains with the same computational power: each hash it calculates can result in a block for either chain. This allows the computational power securing a major PoW chain to also secure an expansion chain. However, if a miner misbehaves when generating an expansion chain block, it suffers no penalty on the main chain, which remains oblivious. In Aegis such misbehavior is penalized, so nodes are strongly incentivized to follow the protocol.

2.3 Long-Range Attacks and Checkpoints

Long-range attacks are a type of attack where an adversary presents an alternative history, indistinguishable from the true one, to the network. This can be used to double-spend coins, or to prevent the network from reaching consensus. Newcomers to the network are vulnerable to these attacks, as they do not have the full history of the blockchain.

Long-range attacks are possible in both PoW and PoS blockchains. In PoW, an adversary performing a long-range attack is required to hold a majority of the computational power, and use it to mine a longer chain than the honest chain[2]. However, this form of attack is more severe in PoS, as it doesn't require significant computational power, only a majority of the stake at a given time. This is exacerbated by the fact that old nodes, who have already withdrawn their stake, are no longer at risk of being slashed.

Previous work has explored *checkpoints* to prevent long-range attacks. Every several blocks, a block is checkpointed by nodes, in order for newcomers to later retrieve the latest checkpoint to

[2]More specifically, this is called a 51% attack [8].

be able to distinguish between the current chain and a long-range attack. Earlier work considered social consensus [9, 13, 21], where a committee of participants, independent of the blockchain, agrees on the current state of the blockchain and communicates it to others via some external medium. Later, some works utilized a BFT consensus protocol to perform the checkpointing [51, 53]. Afterward, the idea of using a separate proof-of-work blockchain to timestamp checkpoints was introduced [32, 6, 59]. These works suggested injecting a digest of the checkpoint into a secure PoW blockchain as a transaction, thus timestamping the checkpoint. Another related work is Winkle [5], which suggests weighing checkpoints based on the volume of transactions which were place in blocks following the checkpoint.

Similarly, Aegis enjoys protection from long-range attacks due to its checkpoints. But unlike them, even if the expansion chain is stalled, reconfiguration occurs in the primary chain, and Aegis ensures that this will be reflected in the expansion chain. None of the aforementioned solutions can guarantee this property, as they all implement reconfiguration within the expansion chain.

2.4 Chain Combining

Fitzi et al. [27] and Wang et al. [61] combine multiple ledgers to create a ledger with better properties. However, unlike Aegis, all steps are taken within the ledgers, unlike Aegis, whose expansion chain can progress by expansion-chain consensus steps that are not immediately checkpointed.

2.5 Layer-2 Blockchains

Various systems utilize a trusted primary blockchain to validate and secure second-level applications. This includes rollups that increase the primary chain's throughput, and offer lower latency and cost using *optimistic* [31, 3, 47] and *zero-knowledge* [56, 39] methodologies. Gudgeon et al. [30] provide a comprehensive systemization of knowledge. However, while the validity of the operations of those systems is guaranteed by the underlying blockchain, their content is, by and large, approved by centralized services [39, 47]. In contrast to those solutions, Aegis implements a decentralized sequencing mechanism, which can be used by any of those rollups together with their on-chain validation mechanisms.

Other solutions include so-called *sidechains* and *payment channels*. Sidechains (e.g., [48, 29]) are independent blockchains with mechanisms to transact with a primary chain, but their security is independent of the primary chain, unlike the expansion chain in Aegis. Payment channels are point-to-point channels [49, 24], which can aggregate several payments between two parties. These can also be used to form a payment network, allowing payments between any two parties in the network, without the need for a direct channel between them. But, as each channel is exclusively controlled by its two counterparties, payment channels are not decentralized, like Aegis.

3 Model

The system (§3.1) includes a primary blockchain and a set of nodes. We assume a generic consensus algorithm (§3.2) that the nodes use aiming to form a secondary blockchain (§3.3). Appendix A summarizes the notation.

3.1 Principals and Network

The system comprises an unbounded set of nodes (as in [34, 50]) and a *primary blockchain* (or simply *primary*) denoted by $\underline{\mathcal{L}}$. Time progresses in steps. In each step a node receives messages from previous steps, executes local computations, updates its state, and sends messages.

The primary chain is implemented by a trusted party that maintains a state including token balance for each node and a so-called *staked amount* for each node. Initially, each node has some initial token balance and zero staked tokens. The primary also maintains node-defined *smart contracts* implementing arbitrary automatons. Nodes interact with the primary by issuing transactions that update its state and interact with smart contracts. The primary chain aggregates transactions into *blocks* and extends the blockchain at set intervals. Its state is the result of parsing the series of all transactions. A node cannot guess the hash of a future block except with negligible probability.

A node can issue a transaction to *stake*, i.e., deposit some of their tokens, or *unstake*, i.e., order the withdrawal of tokens she had previously staked. Unstaking completes Δ_{active} steps after the order transaction is placed in the chain. While a node's tokens are staked, the node is considered *active*. During this period, a node can either follow the protocol (and be considered *correct*) or misbehave arbitrarily. Correct nodes adhere to the protocol while they are staked, if misbehavior could result in losing their stake. However, once they unstake, they are no longer bound by the protocol and can behave arbitrarily without risk of losing their stake.

The set of nodes that have staked but not yet unstaked after a specific block is called a *committee*; until the time any of them completed unstaking, the committee is *active*. In all committees, the ratio of stake held by correct nodes is assumed to be greater than a constant threshold, denoted by α.

Communication among nodes is reliable and partially synchronous: There exists a time t_{GST}, unknown to the nodes, and message delivery is only bounded after it. That is, if a message is sent at time t then it arrives at all nodes by time $\max(t, t_{GST}) + \Delta_{prop}$. In particular, blocks published by the nodes are delivered to all nodes, even those that stake much later.[3]

After t_{GST} the set of active nodes stabilizes, no one stakes or unstakes.

Communication with the primary chain is synchronous: All nodes can observe the current state of the primary chain and issue transactions that are added to it within a bounded time $\Delta_{\underline{\mathcal{L}}\text{-}write}$. This assumption is stronger than the one for internode communication, but it is more reasonable in this case because we rely on an already established primary chain. In practice, this assumption is met by choosing conservative bounds for the time it takes to write to the primary chain.

3.2 Consensus Algorithm

We assume we are given an asynchronous (single-instance) Byzantine fault tolerant consensus protocol (e.g., [12, 15]) implemented in a function *consensusStep*. Called by a process, *consensusStep* takes steps in the protocol, maintaining a local state as necessary. It is executed by a weighted set of nodes $\mathcal{N}$, given as a parameter. (We will later use the staked amount as the weight, thus $\mathcal{N}$ will constitute a committee.) It takes a consensus instance ID, which is a pair of the set of nodes $\mathcal{N}$ and a nonce *id* to distinguish different instances. It also takes a block b the caller wishes to propose. It returns either a block if it reached a decision or $\perp$ otherwise: $\mathit{consensusStep}(b; \mathcal{N}, id) \to \mathit{block}$ or $\perp$.

If starting at t_0, the nodes in $\mathcal{N}$ call *consensusStep* in every step $t \geq t_0$ with the same $(\mathcal{N}, id)$ and the aggregate weight of correct nodes (those who follow the protocol) is greater than α, then the following guarantees hold.

[3] In practice [10], this network is implemented by the nodes' peer-to-peer network, where nodes keep all published blocks available.

Agreement For all correct nodes i, j (maybe $i = j$) and times t, t' (maybe $t = t'$), if node i's *consensusStep* call at step $t > t_0$ with ID $(\mathcal{N}, id)$ returns $b \neq \perp$ and node j's *consensusStep* call at step $t' > t_0$ with ID $(\mathcal{N}, id)$ returns $b' \neq \perp$, then $b = b'$.

Validity If all nodes in $\mathcal{N}$ are correct and they all execute *consensusStep* with the same ID and the same proposal b, then their *consensusStep* call never returns a non-$\perp$ value $b' \neq b$.

Termination For all correct nodes i and times $t > \max\{t_0, t_{GST}\} + \Delta_{consensus}$, node i's *consensusStep* call with ID $(\mathcal{N}, id)$ returns a non-$\perp$ value.

The agreement property implies that a block, which a correct node has voted for, cannot be overturned. Protocols that do not satisfy this property are out of scope.

We require the given BFT consensus protocol to include two additional functions. The function *consensusValidate* takes a block and an ID (including the set $\mathcal{N}$) and returns True if and only if the block is the result of those nodes, with a correct ratio above α, running *consensusStep*. (In practice this can be implemented with cryptographic signatures.)

Finally, the protocol supports *forensics*, that is, it allows identifying misbehavior of the nodes [54, 20]. Specifically, if the number of Byzantine nodes is larger than the bound, they can cause an agreement violation. In such a case, the correct nodes can publish data allowing them to identify misbehaving parties.

3.3 Goal

The goal of the system is to form an expansion blockchain, denoted by $\mathcal{L}$. Each node i has an input value v_i for each position in the ledger. We do not consider the content of this blockchain, which could serve as a decentralized sequencer, oracle, etc. (see §2). We only consider its blocks, which are maintained by the nodes as local vectors. If a node stores a block in a position k in its local vector we say it *logs* this block. Note that this is a chain, not a single-instance consensus as in Section 3.2. The expansion blockchain should achieve the following (referring to expansion-chain blocks and positions):

Agreement If a correct node i logs a block b in position k at time t and a correct node j logs block b' in position k at time t' (maybe $i = j$ and maybe $t = t'$) then $b = b'$.

Validity If all nodes are correct and their inputs for all positions $v_1, v_2, \ldots$ are the same, then in all positions k no node logs a block with a different value $u_k \neq v_k$.

Progress There exists a time $t > t_{GST}$ such that after t, each correct node logs new blocks at a rate of at least $1/(\Delta_{consensus} + \Delta_{prop})$.

When it is not clear from context, we refer to the blockchain properties as blockchain-agreement etc., and the consensus properties as consensus-agreement etc.

4 Aegis

We overview the protocol design (§4.1), then present the Aegis blockchain structure (§4.2) and the primary-chain contract (§4.3). We then describe the peer-to-peer protocol Aegis nodes execute to form the blockchain (§4.4), and discuss the execution of the forensics procedure to identify and penalize misbehaving nodes (§4.5).

4.1 Protocol Outline

Aegis includes two components: A protocol that Aegis nodes execute and a smart contract that the primary chain does. Roughly, the protocol executes a series of consensus instances logging blocks consecutively. Each block b contains a reference (hash) to its *parent*. The blocks thus form a tree, rooted at an agreed-upon *Genesis* block, with each having a height $height(b)$: the number of edges on the path from b to the Genesis block.

Each block also contains a reference to a primary block B. The primary block implicitly specifies a committee—the set of nodes with staked tokens at the state of block B, weighted by the staked amounts. (Nodes must include contact information in the staking transaction, so other nodes would know where to send messages; we omit this detail to simplify the presentation.) This committee is, in the good case, the committee that chooses the block following b. For example, in Figure 1, Aegis block $A2$ references primary block $P4$, so the committee defined by block $P4$ chooses the subsequent Aegis block $A3$.

Occasionally, Aegis nodes *checkpoint* blocks by sending them to the Aegis smart contract on the primary chain. The contract receives the checkpointed block and *accepts* the checkpoint only if its committee is active.

If Aegis could not produce a block for a duration of Δ_{active}—that is, in the bad case—the previous committee might not be active. In this situation, a new committee must be selected to extend the chain. For this, some node issues a *reset* in the primary-chain Aegis contract. The contract would accept the reset only if there were no checkpoints or resets in the last Δ_{active} steps. This initiates a new committee, specified by the block in which the reset occurs. This committee can produce a new block following the previous checkpointed Aegis block using the committee specified by the reset.

Figure 1 illustrates the topology of the Aegis chain $(A1, A2, \dots)$ and the primary chain $(P1, P2, \dots)$. Most Aegis blocks ($A2$–$A6$) are generated by the committee referenced by their ancestor signified by the same color. Some are checkpointed in primary blocks $(P3, P5, P6)$. If a checkpoint cannot be placed in time (and before the first Aegis block), a reset is issued $(P1, P7)$, and the subsequent Aegis block is generated by the committee of the reset block $(A1, A7)$.

We proceed to describe the algorithm in detail.

4.2 Chain Structure

Aegis's data structure includes the Aegis chain and relevant updates in the primary chain.

All nodes start with a single Genesis Aegis block (Algorithm 3 line 2). All Aegis blocks b (apart from Genesis) have a parent Aegis block $parent(b)$ and an Aegis-to-Primary reference to a primary block $ref_{A\to P}(b)$. (For genesis only: $parent(b_{genesis}) = \perp$ and $ref_{A\to P}(b_{genesis}) = \perp$.)

Each Aegis block b should be generated by consensus among the committee it specifies. The committee is specified in one of two ways. First, an aegis block can have a reference to a reset in a primary block, denoted $ref^{reset}_{A\to P}(b)$. If $ref^{reset}_{A\to P}(b) \neq \perp$, the committee is the one specified by the reset. Otherwise, the committee is the one specified in the primary block referenced by b's parent, i.e., $ref_{A\to P}(parent(b))$. The function $stakers(B)$ returns the committee specified by a primary block B. The block should thus be valid according to the *consensusValidate* function with the appropriate committee. The consensus instance ID is the pair consisting of its parent and reset references.

The function *isValid* (Algorithm 1) validates all this—Note that this is only a data structure validation that does not take into account committee activity. The protocol ignores blocks that are invalid as checked by *isValid*.

Algorithm 1: Valid block predicate

function *isValid* $(b, \underline{\mathcal{L}}, \mathcal{B})$
  if b *is Genesis* **then return** True
  if *A checkpoint in* $\underline{\mathcal{L}}$ *conflicts with* b **then return** False
  if $parent(b) \notin \mathcal{B}$ **then return** False
  if $ref_{A\to P}(b) \notin \underline{\mathcal{L}}$ **then return** False
  if $ref_{A\to P}(b)$ *is not a descendant of* $ref_{A\to P}(parent(b))$ **then return** False
  if $\neg isValid(parent(b), \underline{\mathcal{L}}, \mathcal{B})$ **then return** False (Recursively validate parents)
  if $ref^{reset}_{A\to P}(b) \neq \perp$ **then**
    if $ref^{reset}_{A\to P}(b) \notin \underline{\mathcal{L}}$ **then return** False
    if $ref^{reset}_{A\to P}(b)$ *is not a descendant of* $ref_{A\to P}(parent(b))$ **then return** False
    $\mathcal{N} \leftarrow stakers(ref^{reset}_{A\to P}(b))$
  else
    $\mathcal{N} \leftarrow stakers(ref_{A\to P}(b))$
  if $\neg consensusValidate(b, (parent(b), ref^{reset}_{A\to P}(b)), \mathcal{N})$ **then return** False
  return True

Algorithm 2: Aegis primary-ledger contract.

on Receive at time t entry e with pointed-to block b and its parent b'
  if e *is a reset* **then**
    assert no entries later than $t - \Delta_{active}$ (First entry or previous stale)
  else (checkpoint)
    assert $ref_{P\to A}(e) = b$
    assert $parent(b) = b'$
    assert $ref_{A\to P}(b) \in \underline{\mathcal{L}}$
    if $ref^{reset}_{A\to P}(b) \neq \perp$ **then** (b's committee specified by reset)
      assert $ref^{reset}_{A\to P}(b) \in \underline{\mathcal{L}}$
      $B_{reset} \leftarrow \underline{\mathcal{L}}(ref^{reset}_{A\to P}(b))$ (Primary-chain reset entry)
      $\mathcal{N} \leftarrow stakers(B_{reset})$
      $t_0 \leftarrow t_{gen}(B_{reset})$
    else (b's committee specified by previous block)
      $B_{reset} \leftarrow \perp$
      assert $ref_{A\to P}(b') \in \underline{\mathcal{L}}$ (Parent's reference exists)
      $\mathcal{N} \leftarrow stakers(\underline{\mathcal{L}}(ref_{A\to P}(b')))$ (Specified by parent)
      $t_0 \leftarrow t_{gen}(\underline{\mathcal{L}}(ref_{A\to P}(b')))$
    assert $t < t_0 + \Delta_{active}$
    assert $consensusValidate(b, (b', B_{reset}), \mathcal{N})$
    assert $height(b) > height(ref_{P\to A}(\text{previous checkpoint}))$
  accept e (All asserts passed, register checkpoint/reset)

4.3 Primary-Chain Contract

The primary-chain contract (Algorithm 2) allows checkpointing Aegis blocks and initiating resets. It receives *entries* submitted by Aegis nodes asking to register either a checkpoint or a reset.

It registers a reset if there are no entries in the last Δ_{active} steps, indicating one is indeed necessary; no other checks are required.

For a checkpoint, the contract receives an Aegis block to checkpoint along with its parent. It verifies that the block correctly references a primary block earlier than this checkpoint. Similarly, if the Aegis block has a reset reference, the contract verifies it points to a previous primary block. It also verifies that the block's Aegis parent references a previous primary block. The contract then verifies that the committee that created the block is still active by comparing the current time (known to the contract) to the time of the committee's referenced block. It validates the block is the result of the committee's consensus by using the function *consensusValidate* . Finally, it verifies that the block is higher (farther from the Aegis Genesis) than the last checkpointed block. Having passed all these checks, the contract accepts the checkpoint and registers it.

Note 1. *For a practical implementation, to reduce primary-chain overhead, the checkpoint can include only the hash of the Aegis block b, and the following elements with proof they are included*

Algorithm 3a: Aegis algorithm for node i. (part 1/2)

Initialization:
1 $\mathcal{B}_i$: A mapping, initially $\mathcal{B}_i(id(b_{genesis})) = b_{genesis}$, $\forall id \neq id(b_{genesis}) : \mathcal{B}_i(id) = \perp$.
2 $\mathcal{L}_i$: A vector, initially $\mathcal{L}_i(0) = b_{genesis}$, $\forall k > 0 : \mathcal{L}_i(k) = \perp$

Execution at step t: (return immediately if a referenced block is missing)
3 $\mathcal{B}_i \leftarrow \mathcal{B}_i \cup$ blocks received
4 $\underline{\mathcal{L}}_i \leftarrow \underline{\mathcal{L}}_i \| $Primary chain extension receieved (No conflicts by assumption)
5 $B_{last} \leftarrow$ last contract entry in $\underline{\mathcal{L}}_i$ or $\perp$ if none
6 **if** B_{last} *is a reset* **then**
7 **if** $t < t_{gen}(B_{last}) + \Delta_{active} - \Delta_{\underline{\mathcal{L}}\text{-}write}$ **then**
8 $B_{checkpoint} \leftarrow$ latest checkpoint, $\perp$ if none
9 $\mathcal{N} \leftarrow stakers(B_{last})$
10 $t_0 \leftarrow t_{gen}(B_{last})$
11 **else if** $t > t_{gen}(B_{last}) + \Delta_{active}$ **then** (Previous reset stale)
12 issue reset and return
13 **else** (Too close to timeout)
14 return
15 **else if** B_{last} *is a checkpoint* **then**
16 $B_{checkpoint} \leftarrow B_{last}$
17 $\mathcal{N} \leftarrow stakers(\underline{\mathcal{L}}(ref_{A\to P}(\mathcal{B}_i(ref_{P\to A}(B_{checkpoint})))))$ (Committee from checkpointed Aegis block)
18 $t_0 \leftarrow t_{gen}(\underline{\mathcal{L}}(ref_{A\to P}(\mathcal{B}_i(ref_{P\to A}(B_{checkpoint})))))$ (Its generation time)
19 **else** (No block found)
20 issue reset and return (Initialize)
21 **if** $B_{checkpoint} = \perp$ **then** (First non-genesis block, committee by reset)
22 $k \leftarrow 1$
23 $b_{checkpointed} \leftarrow b_{genesis}$
24 $b \leftarrow b_{genesis}$
25 **else**
26 $b_{checkpointed} \leftarrow \mathcal{B}_i(ref_{P\to A}(B_{checkpoint}))$ (We have now found the last checkpoint)
27 **forall** k *from* $|\mathcal{L}| + 1$ *to* $height(b_{checkpointed})$ (Log all blocks up to checkpoint, inclusive)
28 **if** $\exists b \in \mathcal{B}_i$ *s.t.* b *is an ancestor of* $b_{checkpointed}$ *and* $height(b) = k$ **then**
29 $\mathcal{L}_i(k) \leftarrow b$ (Already verified via contract)
30 **else**
31 return (We are missing an ancestor of the checkpoint)

in the block: reference from b to the primary block, reference from b to a reset (could be $\perp$), reference from b to its parent b'. Similarly, for b' we have its reference to a primary block.

Note 2. *Theoretically, neglecting slashing enforcement, a protocol-agnostic append-only log would suffice instead of a smart contract, with the nodes reading it and locally applying the contract logic.*

4.4 Node Protocol

We are now ready to present the protocol executed by each node (Algorithm 3). All nodes start with a single Aegis Genesis block (line 2) and in each time step proceed in three phases, as follows.

4.4.1 Sync to Latest Checkpoint

At the beginning of a step a node first reads the states of the primary and expansion blockchains (lines 3–4). Then, it finds the latest checkpointed block as follows. If the latest primary-chain entry is a checkpoint, it takes this checkpoint; the committee for the next block to log is expected to be the one defined by the checkpointed block, i.e., according to the primary block it references (line 17). Otherwise, if the latest entry is a reset and the committee defined by the reset is still active (line 6), it takes the most recent checkpoint with the reset's committee for the subsequent block. If the reset's committee is inactive, the node issues a new reset and returns, waiting for it to take place. Finally, if there is no primary block entry then this is the first execution of the protocol, so the node issues a reset and returns, waiting for it to take place.

Algorithm 3b: Aegis algorithm for node i. (part 2/2)

32 $k \leftarrow |\mathcal{L}|$ (Equal to $height(b_{checkpointed})$)
33 $b \leftarrow b_{checkpointed}$
34 **if** $t < t_0 + \Delta_{active}$ **then** (Committee for next block still active)
35 Choose $b' \in \mathcal{B}_i$ s.t. $isValid(b', \mathcal{N}, \underline{\mathcal{L}}, \mathcal{B}_i)$ and $parent(b') = b_{checkpointed}$, $\perp$ if none
36 **while** $b' \neq \perp$ **do** (Add blocks by active committees)
37 $b \leftarrow b'$
38 $\mathcal{L}(k) \leftarrow b$
39 $k \leftarrow k + 1$
40 $\mathcal{N} \leftarrow stakers(ref_{A \to P}(b))$
41 Choose $b' \in \mathcal{B}_i$ s.t. $isValid(b', \mathcal{N}, \underline{\mathcal{L}}_i, \mathcal{B}_i)$ and $parent(b') = b$, $\perp$ in none
42 **if** $t < t_0 + \Delta_{active} - 3\Delta_{\underline{\mathcal{L}}\text{-}write}$ **then** (Latest committee has enough time)
43 **if** $b = b_{checkpointed} \wedge B_{last}$ *is reset* **then** (No blocks since last checkpoint and following a reset)
44 $b \leftarrow consensusStep(\text{propose a block}, (\mathcal{L}(|\mathcal{L}| - 1), B_{last}), \mathcal{N})$
45 **else** (Last checkpointed or subsequent block's committee is active)
46 $b \leftarrow consensusStep(\text{propose a block}, (\mathcal{L}(|\mathcal{L}| - 1), \perp), \mathcal{N})$
47 **if** $b \neq \perp$ **then** $\mathcal{L}(|\mathcal{L}|) \leftarrow b$
48 **if** $t = t_{gen}(B_{last}) + \Delta_{active} - 3\Delta_{\underline{\mathcal{L}}\text{-}write}$ *and* $b \neq b_{checkpointed}$ **then** (Deadline to issue checkpoint)
49 issue checkpoint for b and return
50 **if** $t > t_0 + \Delta_{active}$ **then**
51 issue reset and return
(No-op if $t_0 + \Delta_{active} - \Delta_{\underline{\mathcal{L}}\text{-}write} < t < t_0 + \Delta_{active}$)

If the only entry was a reset, namely, there are no checkpointed blocks, then this is the first block after Genesis (lines 21–24). Otherwise, the node goes from the latest block it had confirmed in previous steps (maybe starting from Genesis), logging in its local Aegis chain all ancestors up to the checkpointed block (lines 27–29). If the node is yet to receive any of those, the algorithm stops; the node will wait to receive them in subsequent steps.

4.4.2 Sync Past Checkpoint

Having caught up to the latest checkpoint, the node proceeds to log Aegis blocks finalized by consensus that are yet to be checkpointed. This is only done if the latest checkpointed block's committee is still active (line 34), implying all subsequent committees are still active.

For each block, it adds it to the ledger and updates the next block's committee (lines 34–41). This is done for all Aegis blocks that were generated with currently active committees. Recall that the committee of the first block after the checkpoint is defined by the primary block its parent points to (lines 15–18), or by a reset block if it was produced that way (lines 8–10).

Once this is done, the node is up-to-date, having logged all previous blocks before the last checkpoint and after the checkpoint (with active committees).

4.4.3 Extend the log

If the latest Aegis block specified committee has enough time remaining active, the node tries to extend the chain by taking a consensus protocol step. Specifically, there should be enough time to checkpoint a new block and complete the forensics procedure (line 42). The consensus is identified by the parent and reset blocks (line 44 or 46). If the committee is no longer active, it issues a reset (line 51) to enable progress in future steps. Because the committee defined by the checkpoint is inactive, the contract will accept the reset.

Aegis nodes issue a checkpoint before the committee expires (line 49), which is sufficient for correctness. Note that it could be that just one correct node got the block (line 44 or 46) before the committee expires—the consensus protocol does not guarantee more. Then the perpetuation

of this block relies entirely on this node issuing the checkpoint. But this is guaranteed since this node is correct.

Note 3. *The consensus ID for each Aegis block b includes both its ancestor $ref_{A\to P}(b)$ and its reset reference $ref^{reset}_{A\to P}(b)$, the latter is $\perp$ if the block's committee is specified by its ancestor. This guarantees that the committee could only start the consensus instance when it was active. Without the reset reference, a committee C could be specified by a primary block, later become stale, create a block b while being inactive, then a reset reinstates the same committee, which is now active, allowing it to present b as the result of an active committee.*

4.5 Forensics

Aegis takes advantage of the consensus protocol's forensics support [54, 20] to penalize Byzantine nodes if their number is greater than its threshold and they successfully violate agreement. The forensics procedure encompasses both the contract and the peer-to-peer protocol, we describe it only here for pseudocode readability.

A correct node identifies a violation if it observes two conflicting decided values (by a consensus step, receiving a block, or a checkpoint). In this case, it issues a transaction notifying the contract, which initiates the consensus protocol's forensics procedure. Once the other correct nodes observe the notification in the primary chain, they participate in the procedure, sending the necessary data to the contract. The contract then identifies the Byzantine nodes and penalizes them by slashing (revoking) their stake. The forensics procedure depends on the consensus algorithm's details, and optimization for reducing the primary-chain overhead is outside our scope.

5 Security Overview

The proof is deferred to Appendix B; we briefly review it here. To prove security we must show that Aegis nodes do not register different values at the same height (Agreement), that if all have the same value and none is Byzantine they decide that value (Validity), that after t_{GST} they register values frequently (Progress), and that violation leads to penalties.

We first show by induction that the ancestors of a block generated by an active committee were also generated by active committees. Then, using an indistinguishability argument we show that the guarantees of an existing consensus protocol (with an infinitely-running committee) are preserved during the validity period of an Aegis committee.

The proofs of the three properties follow a similar structure. We prove by induction on the Aegis block number. For each we show the property holds either due to the consensus guarantees, or due to the interaction of the nodes protocol and the primary-chain contract.

6 Conclusion

We present Aegis, a novel expansion blockchain protocol whose nodes can join and leave in a permissionless fashion by staking in a primary chain. Aegis guarantees safety at all times and progress after t_{GST} despite asynchronous communication among its nodes (up to t_{GST}), using committees dynamically specified by a primary-chain smart contract with synchronous access.

While Aegis can be deployed as-is on top of a primary blockchain, there are two practical aspects to consider. Aegis requires primary-chain transactions in intervals close to the time for unstaking Δ_{active}. This implies a tradeoff between short unstaking time and low primary-chain overhead. In terms of performance, Aegis can change committee membership at a lower frequency than every

block. This allows using contemporary BFT state machine replication algorithms (e.g., [64, 28, 7]) that are more efficient than running a one-shot consensus instance per block.

The tools to deploy staking for Aegis are ready at hand, e.g., Avalanche's elastic subnets [38]; direct implementation with a smart contract; and Ethereum's EigenLayer [60], which enables recycling of large stake amounts. Thus, Aegis opens the door for a new generation of truly decentralized expansion blockchains.

References

[1] Cryptocurrency prices, charts and market capitalizations — coinmarketcap. https://coinmarketcap.com/, retrieved May 2024.

[2] Ittai Abraham and Dahlia Malkhi. Bvp: Byzantine vertical paxos. *Proceedings of the Distributed Cryptocurrencies and Consensus Ledger (DCCL), Chicago, IL, USA*, 25, 2016.

[3] Arbitrum. Arbitrum nova. https://arbitrum.io/anytrust, retrieved Sep. 2023.

[4] Pierre-Louis Aublin, Rachid Guerraoui, Nikola Knežević, Vivien Quéma, and Marko Vukolić. The next 700 bft protocols. *ACM Transactions on Computer Systems (TOCS)*, 32(4):1–45, 2015.

[5] Sarah Azouvi, George Danezis, and Valeria Nikolaenko. Winkle: Foiling long-range attacks in proof-of-stake systems. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, 2020.

[6] Sarah Azouvi and Marko Vukolić. Pikachu: Securing PoS blockchains from long-range attacks by checkpointing into bitcoin PoW using taproot. In *Proceedings of the 2022 ACM Workshop on Developments in Consensus*, pages 53–65, 2022.

[7] Kushal Babel, Andrey Chursin, George Danezis, Lefteris Kokoris-Kogias, and Alberto Sonnino. Mysticeti: Low-latency DAG consensus with fast commit path. *arXiv preprint arXiv:2310.14821*, 2023.

[8] Shehar Bano, Alberto Sonnino, Mustafa Al-Bassam, Sarah Azouvi, Patrick McCorry, Sarah Meiklejohn, and George Danezis. Sok: Consensus in the age of blockchains. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, pages 183–198, 2019.

[9] Simon Barber, Xavier Boyen, Elaine Shi, and Ersin Uzun. Bitter to better, how to make Bitcoin a better currency. In *Financial Cryptography and Data Security*, pages 399–414. Springer, Bonaire, 2012.

[10] Bitcoin community. Protocol specification. https://en.bitcoin.it/wiki/Protocol_specification, retrieved Sep. 2013.

[11] Joseph Bonneau, Andrew Miller, Jeremy Clark, Arvind Narayanan, Joshua A. Kroll, and Edward W. Felten. Research perspectives on Bitcoin and second-generation cryptocurrencies. In *Symposium on Security and Privacy*, San Jose, CA, USA, 2015. IEEE.

[12] Gabriel Bracha and Sam Toueg. Asynchronous consensus and broadcast protocols. *Journal of the ACM (JACM)*, 32(4):824–840, 1985.

[13] Vitalik Buterin. Proof of stake: How I learned to love weak subjectivity. https://blog.ethereum.org/2014/11/25/proof-stake-learned-love-weak-subjectivity, November 2014.

[14] Vitalik Buterin. Slasher: A punitive proof-of-stake algorithm. https://blog.ethereum.org/2014/01/15/slasher-a-punitive-proof-of-stake-algorithm/, January 2015.

[15] Ran Canetti and Tal Rabin. Fast asynchronous byzantine agreement with optimal resilience. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 42–51, 1993.

[16] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OsDI*, volume 99, pages 173–186, 1999.

[17] Amedeo Celletti. Random number oracle. https://www.random-oracle.com, retrieved Sep. 2023.

[18] Chainlink. Chainlink 2.0 and the future of decentralized oracle networks. https://research.chain.link/whitepaper-v2.pdf, April 2021. Retrieved May 2024.

[19] Anamika Chauhan, Om Prakash Malviya, Madhav Verma, and Tejinder Singh Mor. Blockchain and scalability. In *2018 IEEE international conference on software quality, reliability and security companion (QRS-C)*, pages 122–128. IEEE, 2018.

[20] Pierre Civit, Seth Gilbert, and Vincent Gramoli. Polygraph: Accountable byzantine agreement. In *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*, pages 403–413. IEEE, 2021.

[21] 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*. Springer, 2019.

[22] Francesco D'Amato, Joachim Neu, Ertem Nusret Tas, and David Tse. No more attacks on proof-of-stake ethereum? *arXiv preprint arXiv:2209.03255*, 2022.

[23] Bernardo David, Peter Gaži, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *Advances in Cryptology–EUROCRYPT*. Springer, 2018.

[24] Christian Decker and Roger Wattenhofer. A fast and scalable payment network with bitcoin duplex micropayment channels. In *Stabilization, Safety, and Security of Distributed Systems*. Springer, 2015.

[25] Cynthia Dwork, Nancy A. Lynch, and Larry J. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, 1988.

[26] eoracle. eoracle. https://eoracle.gitbook.io/, retrieved May 2024.

[27] Matthias Fitzi, Peter Gaži, Aggelos Kiayias, and Alexander Russell. Ledger combiners for fast settlement. In *Theory of Cryptography Conference*, pages 322–352. Springer, 2020.

[28] Rati Gelashvili, Lefteris Kokoris-Kogias, Alberto Sonnino, Alexander Spiegelman, and Zhuolun Xiang. Jolteon and ditto: Network-adaptive efficient consensus with asynchronous fallback. In *International conference on financial cryptography and data security*. Springer, 2022.

[29] Gnosis. Gnosis chain docs. https://docs.gnosischain.com, retrieved Sep. 2023.

[30] Lewis Gudgeon, Pedro Moreno-Sanchez, Stefanie Roos, Patrick McCorry, and Arthur Gervais. Sok: Layer-two blockchain protocols. In *Financial Cryptography and Data Security*. Springer, 2020.

[31] Harry Kalodner, Steven Goldfeder, Xiaoqi Chen, S Matthew Weinberg, and Edward W Felten. Arbitrum: Scalable, private smart contracts. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1353–1370, 2018.

[32] Dimitris Karakostas and Aggelos Kiayias. Securing proof-of-work ledgers via checkpointing. In *2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 1–5. IEEE, 2021.

[33] Mahimna Kelkar, Fan Zhang, Steven Goldfeder, and Ari Juels. Order-fairness for byzantine consensus. In *Advances in Cryptology–CRYPTO*. Springer, August 2020.

[34] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Annual international cryptology conference*, pages 357–388. Springer, 2017.

[35] Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong. Zyzzyva: speculative byzantine fault tolerance. In *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, pages 45–58, 2007.

[36] Petr Kuznetsov and Andrei Tonkikh. Asynchronous reconfiguration with byzantine failures. *Distributed Computing*, 35(6):477–502, 2022.

[37] L2BEAT. Value locked. https://l2beat.com/scaling, retrieved May. 2024.

[38] Ava Labs. Scale with subnets. https://www.avax.network/subnets, retrieved Sep. 2023.

[39] Matter Labs. How decentralized is zksync. https://docs.zksync.io/userdocs/decentralization/#how-decentralized-is-zksync, retrieved Sep. 2023.

[40] Polygon Labs. Change emit. https://github.com/0xPolygon/core-contracts/blob/de71996fbd0b25fbedb630df6bc312e8175e9b7d/contracts/root/staking/CustomSupernetManager.sol#L149, retrieved Sep. 2023.

[41] Polygon Labs. Change vote. https://github.com/0xPolygon/polygon-edge/blob/f2b24e79912802b1b6c8fb3ba1abdbcc9eec94c0/consensus/polybft/state_sync_manager.go#L565, retrieved Sep. 2023.

[42] Polygon Labs. Introducing polygon 2.0: The value layer of the internet. https://polygon.technology/blog/introducing-polygon-2-0-the-value-layer-of-the-internet, retrieved Sep. 2023.

[43] Leslie Lamport, Dahlia Malkhi, and Lidong Zhou. Vertical paxos and primary-backup replication. In *Proceedings of the 28th ACM symposium on Principles of distributed computing*, pages 312–313, 2009.

[44] Leslie Lamport, Dahlia Malkhi, and Lidong Zhou. Reconfiguring a state machine. *ACM SIGACT News*, 41(1), 2010.

[45] Nancy A Lynch. *Distributed algorithms*. Elsevier, 1996.

[46] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. http://www.bitcoin.org/bitcoin.pdf, 2008.

[47] Optimism. Optimism block production. https://community.optimism.io/docs/protocol/2-rollup-protocol/#block-production, retrieved Sep. 2023.

[48] Polygon. Proven scalability on ethereum. https://polygon.technology/polygon-pos, retrieved Sep. 2023.

[49] Joseph Poon and Thaddeus Dryja. The Bitcoin Lightning Network, draft 0.5. http://lightning.network/lightning-network.pdf, February 2015. Retrieved May 2024.

[50] Youer Pu, Ali Farahbakhsh, Lorenzo Alvisi, and Ittay Eyal. Gorilla: Safe permissionless byzantine consensus. In *37th International Symposium on Distributed Computing*, 2023.

[51] Ranvir Rana, Dimitris Karakostas, Sreeram Kannan, Aggelos Kiayias, and Pramod Viswanath. Optimal bootstrapping of pow blockchains. In *Proceedings of the Twenty-Third International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing*, pages 231–240, 2022.

[52] Team Rocket, Maofan Yin, Kevin Sekniqi, Robbert van Renesse, and Emin Gün Sirer. Scalable and probabilistic leaderless bft consensus through metastability. *arXiv preprint arXiv:1906.08936*, 2019.

[53] Suryanarayana Sankagiri, Xuechao Wang, Sreeram Kannan, and Pramod Viswanath. Blockchain cap theorem allows user-dependent adaptivity and finality. In *Financial Cryptography and Data Security*. Springer, March 2021.

[54] Peiyao Sheng, Gerui Wang, Kartik Nayak, Sreeram Kannan, and Pramod Viswanath. BFT protocol forensics. In *Proceedings of the 2021 ACM SIGSAC conference on computer and communications security*, pages 1722–1743, 2021.

[55] Yonatan Sompolinsky, Shai Wyborski, and Aviv Zohar. Phantom ghostdag: a scalable generalization of nakamoto consensus. In *Proceedings of the 3rd ACM Conference on Advances in Financial Technologies*, pages 57–70, 2021.

[56] Starkware. StarkEx High-Level Overview. https://docs.starkware.co/starkex/overview.html, retrieved Sep. 2023.

[57] Selma Steinhoff, Chrysoula Stathakopoulou, Matej Pavlovic, and Marko Vukolić. Bms: Secure decentralized reconfiguration for blockchain and bft systems. *arXiv preprint arXiv:2109.03913*, 2021.

[58] Nick Szabo. Smart contracts: building blocks for digital markets. *EXTROPY: The Journal of Transhumanist Thought,(16)*, 18(2):28, 1996.

[59] Ertem Nusret Tas, David Tse, Fangyu Gai, Sreeram Kannan, Mohammad Ali Maddah-Ali, and Fisher Yu. Bitcoin-enhanced proof-of-stake security: Possibilities and impossibilities. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 126–145. IEEE, 2023.

[60] EigenLayer Team. Eigenlayer: The restaking collective. https://docs.eigenlayer.xyz/overview/whitepaper, retrieved July. 2023.

[61] Xuechao Wang, Peiyao Sheng, Sreeram Kannan, Kartik Nayak, and Pramod Viswanath. Trustboost: Boosting trust among interoperable blockchains. *arXiv preprint arXiv:2210.11571*, 2022.

[62] M Whittaker, N Giridharan, A Szekeres, J Hellerstein, H Howard, F Nawab, and I Stoica. Matchmaker paxos: A reconfigurable consensus protocol. *Journal of Systems Research (JSys)*, 1(1), 2021.

[63] Gavin Wood. Ethereum yellow paper. https://web.archive.org/web/20160820211734/http://gavwood.com/Paper.pdf, 2015.

[64] 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*, pages 347–356, 2019.

[65] Qiheng Zhou, Huawei Huang, Zibin Zheng, and Jing Bian. Solutions to scalability of blockchain: A survey. *Ieee Access*, 8:16440–16455, 2020.

A Notation

Table 1 summarizes key notation used in this paper.

Table 1: Notation

Symbol	**Meaning**
α	Minimum ratio of correct nodes in an active committee
t_{GST}	Global Stabilization Time
Δ_{prop}	Network propagation bound after t_{GST}
Δ_{active}	Time between unstake order and funds withdrawal
$\Delta_{\underline{\mathcal{L}}\text{-}write}$	Time to write a block to the primary chain
$\Delta_{consensus}$	Time to reach consensus after t_{GST}
$H(B) = \hat{B}$	Hash of block B
$\underline{\mathcal{L}}$	Primary ledger
$\underline{\mathcal{L}}(\hat{B})$	Block B if $B \in \underline{\mathcal{L}}$, otherwise $\perp$
$\mathcal{L}$	Aegis ledger
$\mathcal{L}(k)$	Aegis block at height k
$parent(b)$	Parent block of block b
$\mathcal{B}(\hat{b})$	Block b if b is in block set B, otherwise $\perp$
$ref_{A\to P}(b)$	Reference from block b to a primary block
$ref^{reset}_{A\to P}(b)$	Reference from block b to a reset in a primary block
$ref_{P\to A}(B)$	Checkpoint reference from primary block b to an Aegis block
$stakers(B)$	Committee defined by primary block B
$t_{gen}(b)$	Generation time of block b

B Security

After showing that the ancestors of a block generated by an active committee were also generated by active committees (§B.1), we prove Aegis achieves agreement (§B.2), validity (§B.3) and

progress (§B.4), and that it penalizes misbehaving nodes (§B.5). The security guarantees follow (§B.6).

B.1 Active Committee Follows Active Committees

We call a committee active if its nodes' collateral was locked recently enough so it could not be withdrawn yet.

Definition 4 (Active committee)**.** *A weighted set of nodes $\mathcal{N}$ is an* active committee at time t *if there exists a primary-chain block published in the interval $[t - \Delta_{active}, t]$ and those nodes staked collateral up to that block and did not unstake by that block.*

We show that if a block is generated by an active committee then its ancestors were approved by active committees. This is enforced for each block by the committee that generated the block, whether it is defined by a reset or by a previous block whose committee is active.

Lemma 5. *If at time t, a node observes (i.e., receives or generates) a block b such that block b's committee is active at t, then a node (perhaps the same node) receives the parent of b, $parent(b)$, at time t', and $parent(b)$'s committee is active at t'.*

Proof. If a node observes a valid block b, approved by the committee defined by it, there are two options: If the committee is defined by its parent $ref_{A\to P}(parent(b))$, then the committee is active between $t_{gen}(ref_{A\to P}(parent(b)))$ and $t \geq t_{gen}(b)$. If the committee is defined by a reset pointed by $ref^{reset}_{A\to P}(b)$, then the committee is active between $t_{gen}(ref^{reset}_{A\to P}(b))$ and $t \geq t_{gen}(b)$. In both cases, a quorum of the committee nodes approved block b during this interval—recall the block hashes are unpredictable, so the consensus ID is generated on the generation of the primary-ledger block. By the lemma assumption, during this time (up to t) the committee is active; therefore, there is a time $\tilde{t}$ where a correct committee node i confirmed the correctness of $parent(b)$: A node only participates in block generation (lines 44 or 46) after approving its predecessor. Node i logged the parent block b' and validated its consensus in one of three ways: (1) It is the genesis block (line 2), which needs no validation; (2) the parent is checkpointed (line 29), either following a series of resets or not; then b' was approved by an active committee by the time it was checkpointed, as verified by the contract (Algorithm 2 line 18); or (3) the parent is not checkpointed; in this case the node i that logged b only logged b' because the committee of b' is active at $\tilde{t}$ (Algorithm 3 line 34) and the block is valid (lines 35, 41).

In conclusion, the parent block b' was approved by an active committee. □

We next deduce that the ancestors of a block approved by an active committee are all approved by active committees. The proof follows by induction.

Lemma 6. *If at time t a node observes a block b such that block b's committee is active at t then for all ancestors b' of b, there exist a node (perhaps the same node) and time t' such that the node receives b' at t' and the committee of b' is active at t'.*

B.2 Agreement

For agreement we need two more lemmas. First, we ensure steps taken by the nodes are reflected in the primary chain. We show that if the nodes agree on the ledger up to some height, then a subsequent block will be accepted by the primary-chain contract.

Lemma 7. *If there are no two correct nodes that log different blocks at the same height for all indices up to $k-1$ and a node receives a valid block with index k at time t, then a block of height at least k is checkpointed by step $t + \Delta_{active}$ and no resets are registered between t and $t + \Delta_{active}$.*

Note that the lemma does not state that the registered checkpoint is for a descendant of the valid block at height k received.

Proof. The block's committee is specified by the primary block its parent points to or by a reset on $\underline{\mathcal{L}}$. Denote that $\underline{\mathcal{L}}$ block by B. In both cases, some correct node i in that committee observes block b. If within Δ_{active} steps of B there is no checkpoint, then node i issues a checkpoint (Algorithm 3 line 49). We should show the checkpoint is accepted by the contract. There cannot be a reset before Δ_{active} has passed, enforced by the contract (line 3). If the contract accepts checkpoints for ancestors of b then b's checkpoint will still be accepted when it arrives. Suppose the contract accepted checkpoints of blocks at height $k-1$ or earlier that are not ancestors of b. The contract only accepts blocks approved by an active committee (Algorithm 2 line 19). So a correct node in a quorum of an active committee approved a block at a height smaller than k, contradicting the lemma assumption.

Thus, the only case where the checkpoint for b is not accepted is if a checkpoint for a block at height k or more was already accepted, completing the proof. □

We also show that consensus agreement applies to decisions taken by active committees.

Lemma 8 (Active committee agreement)**.** *In an execution of Aegis, an active committee does not decide different values for the same consensus instance.*

Proof. Consider an execution σ of Aegis where two correct nodes in a committee active in the time range $[t_0, t_f]$ decide in steps $t_0 \leq t_1, t_2, \leq t_f$ two values v_1 and v_2 (one each) for the same consensus instance.

Let σ' be an execution with the same prefix as Aegis up to t_1, and extended such that the committee remains active forever (its nodes never unstake), as in the consensus protocol assumptions. By the agreement property of the consensus protocol, the two nodes cannot decide distinct values.

Since at t_1 the nodes cannot distinguish between σ and σ', the nodes take the same steps in σ up to t_1, so $v_1 = v_2$. □

We are now ready to prove agreement.

Proposition 9 (Agreement)**.** *Two correct nodes do not log different blocks at the same height.*

Proof. We prove by induction on the block number k.

Base. For $k = 0$ all correct nodes at all times log the Genesis block (Algorithm 3 line 2; $\mathcal{L}(0)$ is never updated elsewhere). The contract initially has no checkpoints.

Assumption. For $k > 0$, assume that the claim holds for $k-1$. That is, for all nodes i and j (maybe $i = j$) and times $t_1 \leq t_2$, if node i logs a block b at height $k' \leq k-1$ at time t_1 ($\mathcal{L}_i^{t_1}(k') \neq \perp$) and node j logs block b' at height k' at time t_2 ($\mathcal{L}_j^{t_2}(k') \neq \perp$), then $\mathcal{L}_i^{t_1}(k') = \mathcal{L}_j^{t_2}(k')$. Also assume there is no checkpoint for a block b at a height $k' \leq k-1$ and a node i that logs a block $b' \neq b$ at height k'.

Step. Now we prove for k. Assume for contradiction that there exist nodes i and j (maybe $i = j$) and times $t_1 \leq t_2$ such that node i logs a block b in position k at time t_1 and a node j logs block $b' \neq b$ in position k at time t_2: $\mathcal{L}_i^{t_1}(k) = b \neq b' = \mathcal{L}_j^{t_2}(k)$.

The content of the ledger in positive positions is determined due to checkpoints (line 29) or consensus (lines 38 and 47).

If a node logs a block b in position k due to consensus (lines 38 or 47), then it is because it received active committee consensus message for b (activity period verified in line 34). The committee is defined by b's parent or specified by a reset.

If a node logs a block b in position k due to a checkpoint (line 29), then it is because b is an ancestor of a checkpointed block $\hat{b}$. The checkpointed block $\hat{b}$ is due to an active committee verified by the contract (Algorithm 2 line 19). Therefore (Lemma 6), all ancestors of $\hat{b}$ are also approved by active committees. In particular, block b is approved by an active committee. That committee is either the one defined by the previous block, or specified by a reset.

In summary, both blocks b and b' are created by active committees, each either defined by a reset or by the committee referenced by the previous block.

We now consider each of the possible cases. By the induction assumption, both i and j agree on the previous block. If the committees of both blocks are specified by the previous Aegis block, since by assumption they agree on this prior block, that committee decided conflicting values, contradicting the consensus protocol agreement property (Lemma 8).

Next, if the committees of both b and b' are specified by reset blocks, we show it is the same reset block. Assume for contradiction that block b (without loss of generality) is confirmed by the first reset and that b' is confirmed by a later reset. Since block b is approved by the first reset committee (reset at time t_{reset}), by Lemma 7 a block of height at least k is issued a checkpoint before the reset committee becomes stale, with the checkpoint being written by time $t_{reset} + \Delta_{active}$ for a block of height at least k. The contract will only accept another reset after $t_{reset} + \Delta_{active}$ (Algorithm 2 line 3), so after the checkpoint. A committee following a reset considers the latest checkpoint (Algorithm 3 line 8), so a block b' confirmed by the committee specified by the second reset cannot extend the chain earlier than $k+1$ and cannot have the same height as b. A contradiction.

Finally, consider the case where the committee of b (without loss of generality) is specified by its parent and that of b' is specified by a reset. There are two possible cases. In the first case, $ref_{A\to P}(b_{k-1})$ is before $ref^{reset}_{A\to P}(b')$ in the primary chain. All nodes agree on all blocks up to $k-1$ and an active node observed block b with height k, therefore (Lemma 7) a block at height k or more is checkpointed before the contract accepts any reset. So a block at height k is checkpointed before the reset defining the committee of block b'. An Aegis block b' following such a subsequent reset cannot result in a block at height k, a contradiction.

The alternative case is that $ref^{reset}_{A\to P}(b')$ is before $ref_{A\to P}(b_{k-1})$. In this case block b' is not valid due to the order of its parent reference and reset (Algorithm 1 line 10), so no node would log it. Again, a contradiction.

Having reviewed all cases and reached contradictions, we conclude that no two nodes log different blocks at the same height. Since the contract will only checkpoint a block at height k if its committee is active, a checkpoint can only be produced if an active committee decides a different block at height k. We have shown this is impossible, thus a checkpointed block is not different from a valid block received by a node at height k.

This completes the induction step and thus the proof. □

B.3 Validity

To prove validity, we first note that the consensus protocol guarantees validity of an active committee decision.

Lemma 10 (Active committee validity)**.** *In an execution of Aegis, if all committee members of an active committee have the same input value, and Aegis executes the consensus protocol of this committee, then no correct node outputs a different value.*

The proof is similar to that of Lemma 8. We are now ready to prove validity.

Proposition 11 (Validity)**.** *Aegis achieves validity.*

Proof. We prove by induction on the block number k.

Base. For $k = 0$ all correct nodes at all times log the Genesis block (Algorithm 3 line 2; $\mathcal{L}(0)$ is never updated elsewhere). The contract initially has no checkpoints.

Assumption. For $k > 0$, assume that the claim holds for $k - 1$. That is, for all nodes i and times t, if node i logs a block b at height $k' \leq k - 1$ at time t_1 ($\mathcal{L}_i^{t_1}(k') \neq \perp$) and all nodes are correct with input $v_{k'}$ for height k', then $\mathcal{L}_i^{t_1}(k') = v_{k'}$. Also assume there is no checkpoint for a block b at a height $k' \leq k - 1$ with value different from $v_{k'}$.

Step. Now we prove for k. Assume for contradiction that there exists a nodes i and time t such that node i logs a block b in position k at time t and $\mathcal{L}_i^{t_1}(k) = b \neq v_k$.

The content of the ledger in positive positions is determined due to checkpoints (line 29) or consensus (lines 38 and 47).

If a node logs a block b in position k due to consensus (lines 38 or 47), then it is because it received active committee consensus message for b (activity period verified in line 34). The committee is defined by b's parent or specified by a reset.

If a node logs a block b in position k due to a checkpoint (line 29), then it is because b is an ancestor of a checkpointed block $\hat{b}$. The checkpointed block $\hat{b}$ is due to an active committee verified by the contract (Algorithm 2 line 19). Therefore (Lemma 6), all ancestors of $\hat{b}$ are also approved by active committees. In particular, block b is approved by an active committee. That committee is either the one defined by the previous block, or specified by a reset.

In both cases, that committee decided on an invalid value, contradicting the consensus protocol agreement property (Lemma 10).

This completes the induction step and thus the proof. □

B.4 Progress

Finally, we show Aegis extends the ledger after t_{GST}. To prove progress, we first show that consensus termination applies to active Aegis committees after t_{GST}.

Lemma 12 (Active committee termination)**.** *In an execution of Aegis, after t_{GST}, a node in an active committee decides within $\Delta_{consensus}$ steps.*

Proof. Consider an execution σ of Aegis where a node in an active committee participates in a consensus instance in a step $t \geq t_{GST}$.

Let σ' be an execution with the same prefix as Aegis up to $t + \Delta_{consensus}$, and extended such that the committee remains active forever (its nodes never unstake), as in the consensus protocol assumptions. By the termination property of the consensus protocol, the node decides by $t + \Delta_{consensus}$.

Since at $t + \Delta_{consensus}$ the nodes cannot distinguish between σ and σ', the nodes take the same steps in σ up to $t + \Delta_{consensus}$, so the node decides in σ by $t + \Delta_{consensus}$ in σ. □

Proposition 13 (Progress)**.** *Aegis guarantees progress.*

Proof. A node adds a block to the Aegis ledger either since its descendant is checkpointed (Algorithm 3 line 29) or since it is approved by an active committee (Algorithm 3 line 38 or 47). In both cases, the result is that within at most Δ_{active} steps it is checkpointed in the primary chain and seen by all correct nodes (Algorithm 3 line 49).

If at time t the Aegis checkpoint is later than $t - \Delta_{active} + \Delta_{\underline{\mathcal{L}}\text{-}write}$, the nodes take consensus steps and decide within $\Delta_{consensus}$ steps (Algorithm 3 line 46, Lemma 12). However, if the consensus runs for too long, and its result might not be checkpointed in time, it is abandoned (Algorithm 3 line 51). Then no progress can be made until Δ_{active} after the last checkpoint/reset and a new committee is instated with a new reset (line 51). The reset is registered within $\Delta_{\underline{\mathcal{L}}\text{-}write}$ steps and a new committee reaches consensus within $\Delta_{consensus}$ (Lemma 12), guaranteed after t_{GST} because $\Delta_{consensus} < \Delta_{active} - \Delta_{\underline{\mathcal{L}}\text{-}write}$.

From that point on, a new consensus is reached within $\Delta_{consensus}$ steps (Lemma 12) and propagated to all correct nodes within Δ_{prop}, at which point a new consensus instance is started. The interval between blocks is thus bounded by $\Delta_{consensus} + \Delta_{prop}$, as required for progress. □

Note. Independent Proof-of-Stake chains are vulnerable to *long-range attacks*, where a deprecated committee forms an alternative history long after it unstaked. Joining nodes cannot distinguish between real history and the forged one. Aegis is not vulnerable to this attack due to the security of the primary chain.

B.5 Slashing

Enforcing the desired behavior of correct nodes while they are active necessitates penalizing them if they successfully violate safety (Agreement or Validity). We show Aegis achieves this.

Proposition 14 (Penalty)**.** *If a node i belongs to an active committee that violates the consensus safety, it will be penalized.*

Proof. With even a single Byzantine node in the committee, validity vacuously holds. Thus, the only potential safety violation is for agreement, resulting in two distinct nodes deciding different decisions. If a node decides a value at height k, then this or another node will issue a checkpoint for a subsequent block at height at least k within $\Delta_{active} - 3\Delta_{\underline{\mathcal{L}}\text{-}write}$ steps (Algorithm 3 line 49). Thus, if a node i decides a value v_i^k and a node j decides a value $v_j^k \neq v_i^k$, then one of them will find out once the first checkpoint is written to the primary chain or earlier through direct communication of a consensus step (line 44 or line 46) or from received blocks (line 38). Once a node detects such an agreement violation, it issues a forensics procedure by publishing the necessary information to the blockchain. This process is completed within $\Delta_{\underline{\mathcal{L}}\text{-}write}$ steps, at which point its counterpart does the same, completing the on-chain forensics process while the committee of the violating nodes is still active. Thus, the contract can penalize the stake, which is yet to be withdrawn. □

B.6 Aegis Security

The conclusion follows directly from propositions 9, 11, and 13, since nodes are penalized (Lemma 14) for violating safety.

Theorem 15. *Aegis ensures Agreement, validity, and progress.*