High-Throughput Blockchain Consensus under Realistic Network Assumptions

[For internal review - not for public distribution - 30.04.2024]

Sandro Coretti
IOG
sandro.coretti@iohk.io

Matthias Fitzi
IOG
matthias.fitzi@iohk.io

Aggelos Kiayias
University of Edinburgh and IOG
akiayias@inf.ed.ac.uk

Giorgos Panagiotakos
IOG
giorgos.panagiotakos@iohk.io

Alexander Russell
University of Connecticut and IOG
acr@cse.uconn.edu

ABSTRACT

In blockchain consensus, messages are validated based on *proof-of-X*, where X stands for some underlying resource such as work (PoW) or stake (PoS). Protocol message validation via such "resource lottery" allows the following adverse behaviors that are seemingly antithetical to high throughput operation: (I) *Protocol bursts*, occur when large volumes of valid protocol messages are withheld by the adversary and released altogether with the intention of emaciating the bandwidth available to honest parties. (II) *Message replays and equivocations* take place when adversary exploits PoX outcomes and hiding behind the pseudonymity of the network spams honest protocol participants with a message rate much higher than its designated PoX rate, crippling in this way the ability of honest parties to communicate effectively. Reconciling these shortcomings with high bandwidth blockchain operation has so far being unsuccessful: prior work either shies away from utilizing all available bandwidth or if it does, it introduces unrealistic network assumptions that "abstract away" the above adversarial behaviors.

We put forward *Leios*, a blockchain protocol overlay that can transform any underlying low throughput base protocol to a high throughput blockchain achieving $(1-\delta)$ of the optimal capacity for any $\delta > 0$. Our protocol combines a number of techniques that may be of independent interest and reconcile the above shortcomings with high throughput without sacrificing on settlement time — Leios impacts settlement time by only a constant number of rounds. Combining Leios with an underlying dynamic availability protocol, either in the PoS or PoW setting, we obtain for the first time a near optimal throughput permissionless "layer-1" blockchain protocol that is proven secure under realistic network assumptions.

1 INTRODUCTION

Permissionless blockchain protocols, beginning with Bitcoin [29], allow a continuously evolving set of mutually distrusting parties to agree on an ever-growing ledger of transactions as long as the majority of a certain underlying resource—e.g., computation, memory, stake—is controlled by the honest parties. These protocols are generally leader-based, i.e., new leaders are repeatedly elected to propose new transaction-carrying blocks to be added to the ledger. Leader election is implemented by a resource-based lottery wherein the probability to win is proportional to a party's relative portion of the overall resources dedicated to the system.

Towards reaching agreement on the ledger, new blocks must be confirmed—which is achieved either by having (a) future leaders build on them and thereby vote for them implicitly or (b) a committee (also elected via the resource-based lottery) vote for them explicitly. Protocols that follow (a) are generally known as "Nakamoto-style" protocols (cf. [29]), in reference to the author of the original Bitcoin paper, whereas protocols that take approach (b) are referred to as "BFT-style" protocols (e.g., [22]), acknowledging the fact that they employ techniques from Byzantine fault-tolerant computing, a research area that predates blockchain systems by several decades.

An important characteristic of such protocols is how well they scale, where the most common scaling metrics are throughput (i.e., the amount of transaction data confirmed per unit of time), memory usage, and also confirmation latency (i.e., the time between a transaction entering the system and it appearing in the ledger). Throughput, in particular, has long been a prominent challenge for blockchain algorithms (cf. [5]) and still lacks a satisfactory solution.

Challenges for High Throughput. Traditionally, blockchain consensus protocols were analyzed in the so-called Δ-*delay model*, in which it was assumed that all messages (e.g., blocks, votes) sent by honest parties are delayed by at most Δ time and scheduled at the network egress by the adversary.

However, the Δ-delay model does not capture any bound on the capacity of the network: the number of messages in transit is unbounded and all messages are diffused subject to the same upper bound for delivery, regardless of their length. Therefore, this networking abstraction is only reasonable for a low throughput setting—it is unrealistic for any form of high-throughput analysis.

Looking towards realistic network models and permissionless blockchain algorithms in the high throughput setting, there are two fundamental challenges that must be tackled:

I. Protocol bursts. When operating protocols at connection capacity, one must be prepared for the adversary to release a large number of valid protocol messages (e.g., blocks) all at the same time—we call this a "protocol burst"— thereby adaptively increasing network load. Note that protocol bursts are comprised of valid

protocol messages that are maliciously concentrated in terms of timing release with the objective to congest the network. Given the permissionless and pseudonymous nature of blockchain systems, it is infeasible to throttle messages according to their specific sources and hence message relaying rules are limited to coarse criteria, such as network-wide freshness conditions (e.g., a message pointing to the hash value of a recent block). This makes protocol bursts an inevitable vulnerability for permissionless blockchains that protocols must somehow tolerate. Of course, the obvious countermeasure to protocol bursts is to leave sufficient bandwidth unused so that bursts can be absorbed in the worst-case; while effective, this countermeasure is completely antithetical to achieving high throughput and as a result a different mitigation approach is needed.

II. Message replays and equivocations. In the permissionless setting, protocols propagate messages based on ephemeral opportunities that enable communication based on some underlying resource. Specifically, messages are accompanied by a "Proof-of-X," (PoX) where X is the underlying resource such as work (PoW) or stake (PoS). From a bandwidth preservation perspective, it is imperative to restrict the message propagation opportunities of a participant so that they correspond directly to the PoXs it wins (which are limited by resource assumptions); this intuitively means that the rate that a party is capable of generating PoXs should also bound the rate that is capable of transmitting valid messages.

In the PoW setting, assuming hash functions are modeled as Random Oracles (RO), it is possible to show that a PoW is non-malleably tied to a particular message and hence the only possible reuse of a PoW success is a straightforward replay of a previous protocol message. Message replays can be easily contained in a gossipping network (by e.g., keeping a database of hashes for at least as long as a PoW remains fresh). In a PoS setting on the other hand, an attacker may create so-called *equivocations*, i.e., multiple versions of the same protocol message (achieved via "double signing"), and as a result such simplistic replay protection is completely ineffective. It is worth noting that message replays and equivocations are adversarial behaviors that go beyond protocol bursts since the adversary can send a number of messages potentially much higher than what might be justified in a valid protocol execution that takes place as part of a protocol burst.

Contributions. We put forth a new blockchain protocol, *Leios*, which acts as a high-throughput overlay given a black-box underlying low throughput blockchain protocol. The resulting protocol achieves a $(1-\delta)$-fraction of the optimal throughput for any $\delta > 0$, assuming large enough network capacity, and is analyzed in a realistic network model motivated by how real-world gossip protocols operate. In this way, Leios is the first permissionless "layer-1" protocol with near-optimal throughput that, compared to prior work, operates over a realistic network model in which protocol bursts as well as replays and equivocations must be mitigated. We achieve our result by suitably combining the following techniques:
(i) *Bandwidth driven concurent block generation.* In order to fully take advantage of available bandwidth, the protocol has validators concurrently processing transactions, computing results, and packaging the outcomes in so called "input-blocks" (IB) that are disseminated in the gossip network. Such IBs are included *by reference* in the underlying low-bandwidth blockchain to facilitate serialization and the merging of their payloads into a unified ledger.
(ii) *Proofs of data availability.* The decoupling of consensus payloads from the underlying blockchain protocol leads to the challenge of missing payloads that may be accidentally—or even maliciously—omitted while the underlying blockchain protocol is invoked to settle the IBs. To mitigate this issue, we only include input blocks for which there is sufficient evidence that the underlying payload is available and correctly structured. (Our technique can be thought of as "voting" for data availability). A first challenge that arises in this context is that the IB production rate is at such a high level that it will be infeasible to vote for each IB individually. To mitigate this issue, we use *endorsement blocks* (EBs), which collect multiple IBs and propose them for a data availability proof. Endorsing proceeds in two stages to ensure the production of at least one *certified* EB that covers all honestly generated IBs, and hence the rate of IBs referenced by the underlying ledger matches the proportion of honest parties that are active in the network.
(iii) *Pipelined protocol architecture.* Given the number of stages that are needed for IBs and EBs and their associated votes to propagate in the network and form the needed certificates, we adopt a pipelined architecture that ensures the IB concurrent block generation continues uninterruptedly. As a result, no bandwidth is wasted by the protocol while votes and endorsements are collected by the protocol participants.
(iv) *Freshest first message delivery.* The protocol as described offers multiple opportunities to send messages opening up the possibility for protocol bursts. We mitigate this by adopting a "freshest first" message delivery policy. Using cryptographic techniques, we associate message headers with reliable timestamps (e.g., in the PoS setting, we can use the current "slot" timestamp and a verifiable random function (VRF) as part of generating the PoS). Subsequently, nodes will gossip all headers but favor downloading message bodies that have the most recent timestamps. Observe that this rule is easy to implement at the network layer and furthermore curbs the ability of the adversary to exploit protocol bursts. Indeed, while adversarial bursts will be queued for delivery, honest parties' messages can only suffer a limited delay due to the burst, which is proportional to the number of messages in the burst that have a preceding timestamp to the timestamp of the honest message.
(v) *Proofs of equivocation.* Given that our freshest-first delivery mechanism is VRF-based, it opens up the opportunity for equivocations: an adversary can magnify the number of messages it can send arbitrarily by reusing the same VRF opportunity. When the headers of two such messages are received by a party, they can immediately be identified as equivocations: while the payloads may be different, the headers correspond to the same VRF calculation on the same timestamp and public random value. As a result any receiving party can form a *proof of equivocation* that can be propagated to other parties. We demonstrate how to leverage our protocol structure and the communication of such proofs of equivocation to ensure that the impact of equivocations is negligible: at most two message headers are propagated per equivocation and each honest party downloads at most a single block body per IB.

Properties of Leios. For ease of presentation we present two variants of Leios: a simplified one, which is useful to understand the approach and techniques involved, and a full one.

Simplified Leios achieves a throughput that is proportional on the ledger quality η of the underlying low-throughput protocol[1] and hence potentially suboptimal. Furthermore it incurs an additional settlement delay that is $O(\log^2 \kappa)$.

We then present the full Leios protocol which achieves near-optimal throughput of $(1-\delta)\alpha_{\mathrm{H}}$, where α_{H} denotes the fraction of honest stake.[2] We show how we can circumvent the impact of possibly low underlying chain quality by having pipelines cross-reference each other. Concretely, let L and λ be parameters; Full Leios achieves for any constant $\delta > 0$,

- a throughput of $(1-\delta)\alpha_{\mathrm{H}}$ relative to network capacity,
- settlement and liveness guarantees comparable to the base protocol, but with a minimum delay of $\Theta((\delta^{-1}-1)^3 L)$, with both δ, L suitable constants.
- fairness (i.e., the fraction of adversarial blocks is equal to the fraction of corrupted stake).

For convenience, both variants of the protocol are first presented assuming the adversary does not create any equivocations. We then demonstrate how we can exploit the structure of Leios to handle equivocations and ensure they do not impact the protocol.

In our presentation we focus on the PoS setting, but we also briefly explain in Section 7, how the construction can be extended to other resources such as PoW. We note that in the case that the base protocol supports dynamic participation (as, e.g., Nakamoto-style protocols generally do), both Simple and Full Leios remain live in the face of fluctuating participation. During times of low participation, the throughput of Leios gracefully falls back to that of the base protocol, without affecting safety or liveness. As participation improves, the high-throughput properties of Leios take effect anew. In this way, for instance, combining our scheme with Ouroboros [12, 25] one can obtain an optimal throughput PoS protocol that supports dynamic participation.

Related work. Low transaction throughput is inherent to "conventional" blockchain protocols (pure Nakamoto or BFT) since block production is strictly sequential, and a new block must reach the subsequent leader before yet another block can be published. This lag inevitably sacrifices throughput, since the network overall remains underutilized waiting for a single block to propagate to all nodes. To see this in some more detail, in Nakamoto consensus [29] average throughput is bounded by fB, where f is the expected number of blocks per second and B the size of a block in terms of transaction bytes. Based on the security analysis of Bitcoin blockchain, [13, 21], a tight bound for security is $f\Delta < (1/p_A - 1/p_H)$ where Δ is the block propagation delay, p_A the probability that the adversary obtains a PoW and p_H the same probability for the honest parties. Given that $\Delta \geq dB/C$ where d is the diameter of the network and C the capacity of a channel between two nodes in the network in bytes per second, it follows that throughput is bounded by $(1/p_A - 1/p_H) \cdot C/d$. Following a similar reasoning, any leader based protocol (i.e., a protocol where a leader proposes a block of transactions that must be decided afterwards for settlement) must satisfy $f\Delta < 1$ which implies a bound of throughput of C/d. It follows that Nakamoto consensus and even any leader based protocol are suboptimal in terms of their utilization of network capacity.

Scaling Nakamoto consensus. Input-endorsers/fruitchains [25, 33] is a technique originally devised to achieve fairness in Nakamoto-style consensus (i.e., the fraction of blocks a party is able to contribute to the settled blockchain is proportional to their resources), which is provably impossible in pure Nakamoto consensus [16]. It works by producing an additional, concurrent stream of decoupled "input blocks" alongside the conventional Nakamoto protocol, where the input blocks carry the ledger transactions, and the chain orders these transactions by referencing the input blocks.

Bagaria *et al.* [3] insightfully observed that the input-endorsers technique can be effectively applied towards achieving high throughput by keeping the underlying Nakamoto protocol "minimal" (and safe) but pushing the input-block production towards the bandwidth limit. Their resulting *Prism* protocol claims to achieve optimal throughput in a network model where network capacity limits are acknowledged, but network delays are still fixed. Note that this decoupling of consensus blocks from payload blocks in order to allow for optimized efficiency traces back to classic improvements in Byzantine Agreement, cf. [18]. A similar effect was achieved in a more involved way by Fitzi *et al.* [17] and Yu *et al.* [36] by running multiple quasi-independent blockchain protocols in parallel and merging the multiple chains into a single ledger. This approach is also known as *Parallel Chains*.

In terms of modeling the network, early works on Bitcoin analysis [20, 32] assume a synchronous model with bounded delay, typically a parameter Δ and message delivery irrespective of message length. Bagaria *et al.* [3] introduced a network model in which each message m takes time $\Delta := |m|/C + D$ to diffuse, where C is the capacity of the network (in bits per second) and D is a propagation delay (at minimum equal to what is imposed by the speed of light). Furthermore, the protocol proceeds in rounds of length Δ. In order not to overwhelm the network, the data volume to be processed per round must stay below ΔC.

An important downside of all models up to that point was that they do not accomodate a message delivery policy that can be used by honest parties to navigate the large volume of messages during a protocol burst. Without such policy it is entirely possible that honest parties would deplete their bandwidth before getting honest parties' incoming messages on time. To address this important consideration, Neu *et al.* [31] introduced a model that separates message (block) headers from block bodies. To diffuse a block, a party first uploads a block into a hypothetical *cloud* that abstracts the network. The header of the block diffuses with a fixed delay Δ_{hdr}—a realistic assumption since headers are small—while any block body may be downloaded once the header is received, but with a rate bounded by the capacity C. Importantly, nodes can choose what messages to download based on a certain policy, e.g., the "download towards the freshest block" is introduced in [31], where "freshest" here is w.r.t. to a cryptographically verifiable message timestamp. Using this more realistic network model, Neu *et al.* [31]

[1] Intuitively, a chain quality of η means that there is an honest block among any sequence of blocks that was produced during η time slots.

[2] It is easy to see that this level of throughput is essentially optimal since even without the burden of consensus one may at best hope to reach a fraction of α_{H} of throughput utilization, since the adversarial nodes may choose not to contribute.

revisit the Parallel Chains high throughput approach and prove it secure albeit by restricting the security model to static corruptions.

It is worth adding to the above that the model of [31] still allows parties to download block bodies as soon as the corresponding header arrives (i.e., the network acting as a "cloud" service makes block bodies available subject to a throughput constraint). In the real world however, block body propagation is achieved via gossipping and thus block bodies only become available after they spend some time traversing the network where they may encounter other messages that can delay them further subject to the propagation policy of the relaying nodes. This important consideration, missed in [31], is captured by the networking functionality that we introduce.

The commonly overlooked issue of equivocations in Nakamoto protocols was also highlighted by Neu *et al.* [31] who observed safety issues resulting from certain double-signing attacks and suggested how various download rules (typically informally applied in protocol deployments) can be used to prove security formally. Unfortunately, all mitigations impact throughput (or security) as an equivocation adversary may waste honest parties' bandwidth by leading them into pursuing protocol histories that are eventually abandoned or invalid. This problem was identified in a recent paper by Kiffer *et al.* [26] who proposed an equivocation defense that has parties download the bodies of at most one version of each block while achieving longest-chain consensus purely on the chain of headers. This leads to situations in which some parties have downloaded a block that ultimately remains in the longest chain while others have not. They address this by introducing equivocation proofs, which consist of two doubly signed headers and, when included in a later block (before some deadline), result in the contents from the corresponding block being annulled (i.e., they are ignored in the ledger). While equivocation proofs rectify PoS blockchain security, the resulting throughput offered is sub-optimal.

BFT consensus. Danezis *et al.* [10] and Spiegelman *et al.* [34] put forth protocols that can be interpreted as the BFT-style analog of high throughput decoupled ledger consensus operation: input blocks are added to a DAG structure, while a BFT subprotocol is applied on top to agree on a serialization of the DAG blocks. Their protocols also claim to achieve a throughput linear in the network capacity. On the other hand, BFT-style analogs of the Parallel Chains approach that achieves high throughput were given by Stathakopoulou *et al.* [35], Gupta *et al.* [23], and Avarikioti *et al.* [2]. We note that these protocols are all applicable to the permissioned setting; adapting them to the permissionless setting while maintaining their throughput claims is an interesting direction for future work.

Sharding and Layer 2 protocols. Blockchain sharding was introduced by Danezis *et al.* [11] and Croman *et al.* [9]. In contrast to a fully replicated blockchain wherein each full node stores the complete ledger state, a sharded blockchain partitions the ledger state by assigning different committees who each maintain a different part of the ledger. We refer to Avarikioti *et al.* [1] for a formal treatment of scalability and throughput of blockchain sharding. Sharding could potentially enhance transaction throughput beyond the network's capacity by avoiding complete ledger replication across all parties. Instead, specific parties are tasked with maintaining shards, including distributing shard data to the public. Layer-2 (L2) protocols, see e.g., [24, 30] offer a different approach to scaling: they typically centralize off-chain processing and then via incentives and/or zero-knowledge proofs ensure that the off-chain execution is correct. While both sharding and L2 approaches can be fundamental facilitators of further scaling – they address a different problem compared to the question of scaling a blockchain (L1) protocol to its absolute physical limits — which is the focus of our work.

2 PRELIMINARIES

2.1 Network model

The network model in this work is captured by a functionality F_{FFD} ("freshest-first diffusion"), which is motivated by the properties of practical gossip protocols combined with a *freshest-first* diffusion strategy.

Real-world gossip protocols. Gossip protocols commonly use an overlay network in which parties forward new messages to their neighbors. In an adversarial environment, in order to avoid DoS attacks, it is important that (i) the application layer (e.g., consensus) restrict the rate of valid messages sent, and (ii) the network layer use the application layer to verify the validity of messages sent and drop illicit ones. Methods of achieving this include proofs of work (PoW) or proofs of stake (PoS) via verifiable random function (VRFs). Unfortunately, the latter approach, adopted in this work, allows the adversary to equivocate messages, i.e., to send two (or more) different versions of the same protocol message.[3] Thus, a DoS-safe network layer has to suitably manage equivocated messages.

The concrete message-diffusion strategy after which F_{FFD} is modeled is as follows: Messages are split into headers h and bodies b with the property that there is a public function $\mathsf{match}(h, b)$ that tests whether a particular b matches a particular header h; furthermore, it is assumed that it is infeasible to find two bodies that match the same header and that a message ID $\mathsf{msgID}(h)$ (from a totally ordered set) is inferable from the header.[4]

Header diffusion works as follows:

(1) The diffusion of a message is initiated by the sender by first sending its header to all of its neighbors.
(2) When a party receives a new header, it is not directly forwarded but rather output to the environment (which corresponds to the application layer).
(3) When a party receives a header as input from the environment, they forward it to all of their neighbors.

The detour via the environment (in Steps (2)–(3)) "externalizes" message validity, which commonly depends on the state of the application layer. The idea is that an honest party verifies a received message and re-inputs it into the network layer if it is valid, thereby contributing to its diffusion.

In order to protect against equivocations, Step (3) is only executed if the party has not already seen multiple versions of the message. Note, however, that the first equivocation is still forwarded to the neighbors, with the goal of spreading "a proof of equivocation (PoE)" (i.e., two versions of the same protocol message) throughout the network.

Finally, body diffusion works as follows:

[3] This is also known as "doubly signing."

[4] An equivocation is thus a pair of different (valid) messages (h, b) and (h', b') with $\mathsf{msgID}(h) = \mathsf{msgID}(h')$.

(1) The sender of a message—in addition to sending the header to its neighbors—also announces when it is in possession of the corresponding body.
(2) When a party learns of a body announcement corresponding to the *preferred* header, which is the *first* header it has seen for a particular message ID, it downloads that body from the sender of the announcement and in turn announces possession of the body to its neighbors. The party then outputs the body to the environment. Most crucially, the parties prioritize body downloads belonging to headers with higher message IDs, which corresponds to a *freshest-first strategy* (assuming larger message IDs signify newer messages).

The network functionality. Based on the motivation above, F_{FFD} allows for the transmission of messages consisting of a header and a body. Due to their small size, a header is delivered to all parties within a fixed delay

$$\Delta_{\mathsf{hdr}} \text{ after sending}$$

(or after the first honest party has seen them).[5]

The delivery time of the bodies, which are much larger, depends on the congestion in the network, on the diameter d (i.e., the shortest path between any two parties) of the underlying overlay graph, and on the capacity C of the network links. Specifically, a body belonging to a header for which there are no double signs is delivered

$$\tau(d + k) \text{ time after sending}$$

(or after the first honest party has seen it) if at that point there are at most k messages with newer message IDs in the network, where $\tau := |b|/C$ is the time it takes to send a body b over a single "hop" in the network.[6] (Of course, the adversary may cause a message to be delivered at any time before.) The rationale behind this is as follows: if there are no fresher messages impeding the delivery of a particular message m, then m reaches all parties after being in transit for at most τd time, while each newer message causes m to be queued at most once.

Note that for equivocated blocks, the body delivery guarantees are limited: they are only delivered provided that all honest parties have the same preferred header (i.e., saw the same header first). This is due to the fact that in the motivating gossip protocol above, parties only download bodies for their preferred headers, and therefore, global body delivery can only be guaranteed if all honest parties have the same preferred header for an ID mid.

The full description of functionality F_{FFD} is provided in Appendix A. The intuitive guarantees outlined here are sufficient for understanding the main body of this paper.

2.2 Security definitions

Model. In order to capture the security of ledger protocols, we adopt a model providing parties access to a number of hybrid functionalities $\mathcal{F}_i$ reflecting infrastructural assumptions such as a (low-throughput) network model, VRFs, signatures, etc. In particular, it is assumed that there is a key registration functionality that parties can use to distribute keys consistently among each other, as in [12].

[5] In order to make the model realistic, Δ_{hdr} should be chosen to also reflect the diameter.

[6] For simplicity, it is assumed that all bodies have the same size.

This work considers only *synchronous* protocols in which parties have access to a shared clock that proceeds in *slots*. A ledger protocol Π has access to these hybrids and provides the interfaces below to the environment; for simplicity, only PoS protocols are considered here, but the definitions generalize to other protocols based on other resources.

As our extension protocol will ultimately rely on the base protocol to settle protocol metadata rather than low-level transactions, we keep the transactions abstract and refer to them as *payloads*. In contrast, the payloads of the extension protocol itself are conventional ledger transactions.

The interface between the environment and a ledger protocol is as follows:

- Initially, the protocol takes an input (INIT, V), where V is a validation predicate taking a payload vector $\vec{p}$ as input. It replies to this by producing an output $(\textsc{Stake}, \text{SD})$ containing the *stake distribution* $\text{SD} = ((P, s_P))_P$, with $s_P \in [0, 1]$ and satisfying $\sum_P s_P = 1$.
- Once per round, the protocol takes input (FTCH-LDG) and returns $(\text{FTCH-LDG}, L)$, where L is a vector of payloads p.
- Once per round, the protocol takes, from the environment, input $(\text{SUBMIT}, \vec{p})$ for a vector of payloads $\vec{p}$—the payloads the party is told to include in the block as a potential block leader.[7]

The security of the ledger protocol is captured w.r.t. an attacker that may corrupt an arbitrary number of parties as long as the stake corresponding to corrupted parties P—based on the stake distribution SD output initially—remains below $1/2$.[8] Throughout this paper, α_{H} denotes the fraction of honest stake.

Security. Ledger protocols are expected to satisfy the usual two fundamental properties: Persistence and Liveness, where Persistence requires that all parties have consistent views of the ever-growing ledger and Liveness requires that honest payloads make it into the ledger.

In the following, let $L_P^{(t)}$ be the random variable corresponding to the ledger (output via the FTCH-LDG command) as seen by party P at time t. Further, let $\vec{p}_{P,\tau}$ be the vector of payloads input to party P at time τ via SUBMIT.

Definition 2.1 (Persistence with window parameter w). For all parties P, P' and all times $t \le t'$, it holds that (i) $L_P^{(t)}$ is a prefix of $L_P^{(t')}$ and (ii) $L_{P'}^{(t-w)}$ is a prefix of $L_P^{(t)}$.

Definition 2.2 (Liveness with parameters r and u). For all times t, the following holds: if some payload p is input to each honest party during r rounds, i.e., in each round $\tau = t, t+1, \ldots, t+(r-1)$, $p \in \vec{p}_{P,\tau}$ for all honest P, then p will be contained in $L_P^{(t+u)}$ for each honest party P.

The above (standard) liveness definition is satisfied by our extension protocol. For the base protocol a "quality" property, sufficient for our extension protocol, is assumed instead of liveness:

[7] We restrict the environment to only input $\vec{p}$ that satisfy $V(p)$ and to input the same V to all parties.

[8] For simplicity it is tacitly assumed that the protocols considered in this work are well-formed in that they output the same stake distribution to all parties. In particular, it is assumed that protocols have a way for parties to coordinate the initial stake distribution as well as coordinate the key registration process, e.g., as done in [12].

Definition 2.3 (Ledger with quality parameter η and liveness parameter u). For all times t, the following holds: at least one honest payload vector from the η slots $\tau = t, t+1, \ldots, t+(\eta-1)$ will be contained in $L_P^{(s+u)}$, i.e.,

$$\left(\bigcup_{\tau=t}^{t+\eta-1} \bigcup_{P_i \in \mathcal{H}} \vec{p}_{i,\tau} \right) \cap L_P^{(s+u)} \neq \emptyset .$$

The fact that the base protocol is a low-throughput protocol can be modeled by the parameter η in conjunction with a restriction on the size of the payload vectors $\vec{p}$ allowed to be input via SUBMIT.

2.3 Verifiable random functions and lotteries

A *verifiable random function (VRF)*, defined by Micali *et al.* [27], is a cryptographic primitive that allows a party P to create a key pair consisting of a secret evaluation key and a public verification key such that: (a) the public key implicitly defines a function f (b) with the secret key, P can evaluate f at any input x, obtaining an output y and a proof π; (c) given the public key of P, anyone is able to verify, using π, that y is indeed the output corresponding to x; (d) with only the public verification key, outputs of the function cannot be predicted, and in fact appear random.

The above guarantees are abstracted by an idealized functionality F_{vrf} (cf. Figure 6), whose full description is provided in Appendix A. The intuitive guarantees outlined here are sufficient for understanding the main body of this paper.

VRFs can be used to implement a stake-based lottery by evaluating the VRF on a lottery identifier and requiring that the output y be below a certain (stake-dependent) threshold.

3 SIMPLIFIED LEIOS

The Leios protocol is a black-box construction of a high-throughput ledger protocol on top of a low-throughput protocol, the *base protocol* Π_{base}. The base protocol is assumed to satisfy persistence (cf. Definition 2.1) and ledger quality (cf. Definition 2.3). Morevoer, it is assumed that the size of payload vectors supported by Π_{base} is large enough to contain the messages the extension protocol must put on the base ledger.

The bandwidths for the base protocol and the Leios extension are separated: Specifically, the base protocol may require arbitrary hybrid functionalities $\mathcal{F}_i$ to run, including one that models the networking assumptions used by the base protocol. Conversely, functionality F_{FFD} introduced in Section 2.1 is used to model the networking assumptions made by the Leios extension, which is designed to be near-optimal in the bandwidth C allocated to F_{FFD}.

This section presents *Simplified Leios*, a version of Leios that (1) achieves near-optimal throughput that is scaled by the ledger quality η of the base protocol and a somewhat undesirable trade-off between liveness and error probability. In rough terms, for parameters L and λ, Simplified Leios achieves[9]

- a relative (w.r.t. C) throughput of roughly $\frac{\lambda}{\lambda+1}\alpha_{\mathrm{H}} \min(L/\eta, 1)$,
- settlement and liveness guarantees comparable to the base protocol, but with a minimum delay of $O(\lambda^3 L)$,
- fairness, and
- security error $O(t)e^{-\Omega(L)}$, where t is an upper bound on the amount of time the protocol is run.

Section 4 shows an augmented version, *Full Leios*, that achieves near-optimal throughput independent of η and with security error independent of L, meaning liveness can be constant (by choosing constant λ and L) while still allowing for negligible error.

For simplicity, both versions of Leios are first presented assuming that the adversary does not attempt to equivocate any protocol messages. Section 5 shows how to remove this assumption.

3.1 Protocol overview

3.1.1 Input blocks. Leios is based on the concept of *input endorsing* [20, 25], in which parties (run local, VRF-based lotteries to obtain the rights to) produce payload-carrying so-called *input blocks (IBs)*, which are then ordered using an underlying (low-throughput) consensus protocol. While sounding rather straightforward, actually implementing this technique requires considerable care.

The idea is to order IBs by including them in the underlying low-throughput ledger *by reference*. This, however, immediately leads to a data availability issue since the adversary may include references for which there are no corresponding IBs. Such behavior can be thwarted by certifying IB availability via (aggregatable) signatures and modifying the ledger rules of the base protocol to only accept certified IBs.

3.1.2 Endorser blocks. Unfortunately, due to the high rate at which IBs need to be produced in order to achieve large throughput, certifying each IB individually would require too much computational overhead as well as space in the ledger of the (low-throughput) base protocol. To that end, IBs are first collected (by reference) in so-called *endorser blocks (EBs)* (also produced based on local, VRF-based lotteries), which are then certified. The idea is that parties contribute to the certification of an EB if they have seen all IBs referenced therein.

3.1.3 Fairness. A blockchain protocol is *fair* if the fraction of honestly created blocks in the chain corresponds to the actual fraction of honest stake (or whatever other resource underlies the security of the protocol). This extends to the input endorser technique in the sense that the fraction of honest IBs ultimately included via the base protocol should correspond to the fraction of honest stake.

The adversary may negatively affect the fairness of an input-endorser protocol by only referencing adversarially generated IBs in adversarial EBs. For example, if EBs are produced at a rate equal to the capacity of the base ledger, then such an adversarial strategy leads to roughly 75% adversarial IBs in settings where the adversarial stake is only slightly below 50%. It is therefore necessary to produce EBs at a rate high enough such that there is at least one honest EB for every block in the base protocol.

However, at such a high rate, not all EBs can be included in the low-throughput base protocol, and including only a subset of all EBs will not solve the fairness problem since it is unclear which EBs are from honest parties. Leios therefore adopts an approach wherein EBs may also reference other EBs.

3.1.4 Freshest-first delivery. As explained in Section 1, IBs are diffused in a *freshest-first* fashion in order to prevent the attacker from spamming the network by releasing old IBs in bulk, which

[9] Recall that α_{H} denotes the fraction of honest stake.

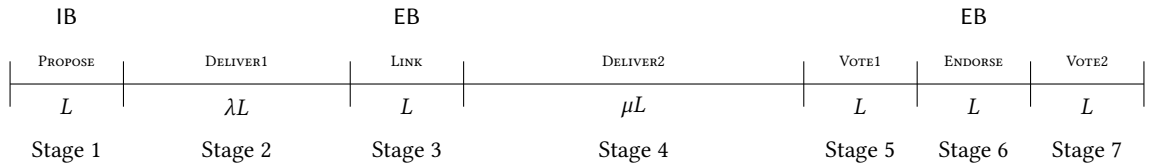

Figure 1: The seven stages of the endorsing pipeline.

is modeled by using F_{FFD} (cf. Section 2.1). This approach comes with its own challenges, however, as special care has to be taken to ensure that IBs arrive before certain deadlines.

3.1.5 The pipeline. The Leios design is a pipelined one. A pipeline instance of Simplified Leios consists of seven stages (cf. Figure 1). Pipeline instances run concurrently, with a new instance beginning every L slots. IBs and EBs are generated at a steady rate throughout the execution, where each IB plays a "proposer" role in the most recent pipeline instance, and each EB plays a "linking" role and an "endorsing" role in two different pipeline instances. The pipeline stages, whose lengths (in slots) are multiples $\{1, \lambda, \mu\}$ of some number L, referred to as the *slice length* (where μ is another parameter defined later), are as follows:

(1) Propose (length L): IBs produced in this stage are the "focus" of the pipeline instance;
(2) Deliver1 (length λL): time for IBs from the propose stage to be diffused;
(3) Link (length L): IBs from the Propose stage are referenced by EBs produced in this stage;
(4) Deliver2 (length μL): time for adversarial IBs referenced by the EBs from the Link stage to diffuse;
(5) Vote1 (length L): parties cast votes for EBs from the Link stage whose referenced IBs have been delivered; if an EB obtains a number of Vote1-votes above a certain threshold, it becomes Vote1-certified;
(6) Endorse (length L): EBs from the Link stage that have garnered a Vote1-certificate are referenced by EBs from this stage;
(7) Vote2 (length L): parties cast votes for EBs from the Endorse stage that satisfy the following property: the EB references (i) Vote1-certified EBs only and (ii) a majority of all Vote1-certified Link-stage EBs; if an EB obtains a number of Vote2-votes above a certain threshold, it becomes Vote2-certified.

3.1.6 Pipeline guarantees. The goal of every pipeline is (a) to produce at least one Vote2-certified EB and (b) that each such EB, via references to Vote1-certified EBs, covers all honest IBs from the Propose stage. Including any such EB per pipeline into the underlying low-throughput protocol will thus ensure liveness of honest input blocks as well as fairness.

Delivering IBs in time for EB inclusion. The (honest) IBs generated in the Propose stage must be delivered to EB producers before the beginning of the Link stage. To that end, Propose is followed by a Deliver1 stage, in which these IBs propagate through the network. However, the Deliver1 stage overlaps with the Propose phases of subsequent pipeline instances, and the IBs produced there take precedence because F_{FFD} delivers IBs in a *freshest-first* manner. In order to guarantee that this interference does not delay the original IBs by too much, the overall IB rate must be such that the Deliver1 stage, which is of length λL, can handle the IB traffic originating during both the Propose and Deliver1 phases. This is ensured by bounding the IB data rate (relative to the capacity of the network) by $\lambda/(\lambda+1)$.[10] Note that therefore, there is a tradeoff between higher throughput (larger λ) and shorter pipeline (smaller λ). The error $e^{-\Omega(L)}$ stems from a Chernoff bound guaranteeing that the number of IBs is close to its expectation.

Delivering IBs in time for first EB voting. Observe that honest EBs produced during Link may only garner enough votes in Vote1 if all the IBs they reference are globally delivered before Vote1 starts. While honest IBs are guaranteed to be globally delivered already before Link (as argued above), the adversary may start diffusing all of its IBs from Propose late, in the worst case just before the end of Deliver1. As a consequence, some honest EB producers may include those adversarial IBs, and therefore a second delivery period, Deliver2, is required for them to be diffused globally before the start of Vote1 as otherwise, these honest EBs may be at risk of being lost due to not receiving enough votes.

In a similar vein to Deliver1 above, the delivery of said adversarial IBs is ensured if the (relative) IB data rate is below $(\mu+1)/(\lambda+\mu+2)$, which essentially guarantees that all the IB traffic of the first four stages fits into the Link and Deliver2 stages. In order not to further restrict the throughput, one requires

$$\frac{\lambda}{\lambda+1} \leq \frac{\mu+1}{\lambda+\mu+2},$$

which is satisfied for $\mu = \Omega(\lambda^2)$.

All honest EBs certified in time. The above guarantees that all honest EBs produced in Link will be (globally seen as) certified before the beginning of the Endorse stage (assuming $L \geq \Delta_{\mathsf{hdr}}$, the header delay in F_{FFD}). They will therefore be referenced by all honest EBs produced in Endorse.

All of the honest Endorse EBs will be Vote2-certified because (i) they only reference Vote1-certified Link EBs and (ii) they all reference a majority of Vote1-certified Link EBs since all honest Link EBs are certified as argued above.

Therefore, by the end of the pipeline, there will be at least one Vote2-certified EB that, via references to Vote1-certified EBs, covers all honest IBs from the Propose stage.

Actual availability of the payload. One final time, the adversary may introduce a delay, this time on the availability of the payload, by adopting the following (worst-case) strategy: just before the end

[10] The diameter of the underlying network (cf. Section 2.1) also plays a role here, but this is ignored in this overview.

of Deliver2, it serves some IBs (that it was supposed to diffuse during Propose), along with an EB that references them, to a small subset of honest parties, just enough to obtain sufficiently many Vote1 votes to get the EB Vote1-certified. The EB will subsequently be picked up by honest EBs in the Endorse phase, and therefore the adversarial IBs end up referenced in the base ledger. However, they still need to be diffused to most honest parties, while also competing with newer IBs.

In the same spirit, these IBs will be delivered after an imaginary Stage 5′ of length νL if the (relative) IB data rate is below $\nu/(2+\lambda+\mu+\nu)$. Again, in order not to further restrict the throughput, one requires

$$\frac{\lambda}{\lambda+1} \leq \frac{\nu}{2+\lambda+\mu+\nu},$$

which is satisfied for $\nu = \Omega(\lambda\mu) = \Omega(\lambda^3)$.

3.1.7 Ledger construction. To extract the ledger, a party can parse the settled part of the base chain, and list all referenced (Vote2-certified) EBs by order of appearance; of this list, the party can extract a list of Link-stage EBs (also in order of appearance), and finally sequence all therein referenced IBs, finally defining an ordered list of ledger transactions.

While deriving the ledger in this fashion, some transactions may become invalid in the context of its preceding transactions and must therefore be dropped (note however that it can still be guaranteed that all included transactions pay a fee, cf. [26]). Morevoer, since the transaction order is not completely known before settlement, generically speaking, all computation required for transaction validation has to be done when a new base block settles that references it. This is especially true in account-based systems, e.g., Ethereum's EVM — even though ideas of concurrent smart contract execution may be able to alleviate some of this cost, cf. [14]. Interestingly, this bottleneck can be avoided in UTxO-based ledgers (e.g., Bitcoin or Cardano, which uses the EUTxO model [8]). In such ledgers, verifying transaction correctness essentially consists of two parts: (1) check that all transaction inputs exist (and are unspent), and (2) that given those inputs, the script outcome is computed correctly. While the first (cheap) part of this computation can only be done once the serialization is complete, the second (expensive) part can be done "speculatively" prior to serialization and in our context during IB validation, i.e., the bulk part of validation is evenly spread over time alongside with IB production, without inducing any computational spikes on the client side.

3.2 Protocol messages

In the following, it will be useful to define a function that returns the slice x slices before the slice that contains some slot s. Specifically, the function returns the interval

$$\mathsf{slice}_L(s,x) := [s', s'+(L-1)],$$

for

$$s' := \lfloor s - xL \rfloor_L,$$

where $\lfloor\cdot\rfloor_L$ rounds down to the nearest multiple of L.

Input blocks. An input block $\mathsf{IB} = (h, b)$ consists of a header h and a body b. The header $h = (s, P, \pi, x, \sigma)$ consists of a slot number s, a stake holder ID P, a lottery proof π, a body hash x, and a signature σ on h (without σ). The body contains a list L_T of ledger transactions.

For the purpose of sending them via F_{FFD}, $\mathsf{msgID}(h) := (s, P)$ and $\mathsf{match}((\cdot,\cdot,\cdot,x,\cdot), b)$ if and only if x is the hash of b.

IBs are created at rate f_{I}, for some parameter f_{I}: given relative stake α, for a given slot, a stakeholder P is assigned the right to produce $\phi_{\mathsf{I}}(\alpha) = \alpha f_{\mathsf{I}}$ input blocks in expectation.[11]

An input block is *valid* if lottery proof π is valid for (s, P) and signature σ is valid for P w.r.t. the public keys in the master index.[12]

Endorser blocks. Endorser blocks $\mathsf{EB} = (s, P, \pi, L_{\mathsf{I}}, L_{\mathsf{E}}, \sigma)$ consist of a slot number s, a stake holder ID P, a lottery proof π, lists L_{I} and L_{E} of IB and EB references, respectively, and a signature σ on EB (without σ). Due to their small size, especially compared to IBs, EBs are modeled as "header-only" for the purpose of sending them via F_{FFD}.

EBs are created at rate f_{E}, for some parameter f_{E}: given relative stake α, for a given slot, a stakeholder P is assigned the right to produce $\phi_{\mathsf{E}}(\alpha) = \alpha f_{\mathsf{E}}$ endorser blocks in expectation. [13]

An EB is *valid* if (a) lottery proof π is valid for (s, P) and signature σ is valid for P w.r.t. the master index, (b) L_{I} only references IBs from $\mathsf{slice}_L(s, \lambda+1)$, and (c) L_{E} only references EBs from $\mathsf{slice}_L(s, \mu+2)$. Note that EB validity can only be determined once all referenced IBs and EBs have been received.

Votes. The voting primitive used provides algorithms for key generation (used to create the corresponding keys in MI), vote creation/verification, as well as certificate creation/verification; these algorithms (except key generation) depend on the stake distribution $((P, s_P))_P$. For efficiency, the voting scheme is sortition-based, which means that the vote creation algorithm only outputs an actual vote for a subset of the parties. A vote/certificate is *valid* if it is accepted by the corresponding verification algorithm. Similarly to EBs, votes are also modeled as "header-only" for the purpose of sending them via F_{FFD}.

On an intuitive level, the voting scheme guarantees that (i) if all honest parties are instructed to vote for a particular message m (i.e., to run the voting algorithm on m), enough votes are generated to create a valid certificate, and (ii) if no honest party votes for m, the adversary cannot create a certificate for m.

It is important to note that, in contrast to IBs and EBs, which are produced over the course of the respective stage, all votes of a given stage (Vote1 and Vote2) are initiated right at the beginning of the stage.[14]

The exact syntax, security properties, as well as an instantiation of the voting scheme (based on ideas from [6, 7, 28]) are detailed in Appendix D, but the intuition provided here is sufficient for understanding the protocol.

Validity. In the following sections, it is tacitly assumed that parties drop all protocol messages that are not valid as defined in this section.

[11] For values of $f_{\mathsf{I}} > 1$, one needs to subdivide each slot into $q > f_{\mathsf{I}}$ subslots, each with $\phi_{\mathsf{I}}(\alpha) = \alpha f_{\mathsf{I}}/q$.

[12] Note that x being the hash of b is already guaranteed by F_{FFD} with the above match predicate.

[13] Similar to footnote 11.

[14] possibly delaying some stage-Vote2 votes by Δ_{hdr} in case a respective EB is produced late in the Endorse stage.

Protocol $\pi_{\mathsf{simpleLeios}}$

Parameters: L: slice length; λ, μ: stage-length multipliers for Deliver1 and Deliver2 phases, respectively; $f_{\mathsf{I}}, f_{\mathsf{E}}$: frequencies of IBs and EBs, respectively (implicit in this figure).

General

Initialization: Upon receiving (INIT, V), store V and create and register the IB/EB lottery and voting keys with the key registration functionality, which as a response outputs the full public key list. The list is used to generate predicate V_{chkCerts}. Send input $(\text{INIT}, V_{\mathsf{chkCerts}})$ to the base protocol; receive $(\text{Stake}, \text{SD})$ from the base protocol and store SD (to be used for the IB/EB lottery and voting).

Network and ledger: At the beginning of each slot:

(1) Fetch new messages (IBs, EBs, and votes) from F_{FFD} and discard invalid ones (cf. Section 3.2).
(2) Compute the ledger by following the EB-(EB-)*-IB references in the base ledger in order and concatenating the payloads in the IBs up to the point of the first IB with an non-downloaded body (if any); sanitize the ledger based on V, as detailed in Section 3.1.7.
(3) Output the resulting ledger upon receiving command FTCH-LDG.

Base chain: In each slot s:

(1) Upon $(\text{SUBMIT}, \text{txs})$, store transactions txs.
(2) Pick an arbitrary Vote2-certified EB EB from $\text{slice}_L(s, 2)$ and send $(\text{SUBMIT}, \text{EB})$ to the base protocol; if there are no such EBs, submit $(\text{SUBMIT}, \text{txs})$ for the (most recently) stored txs.

Protocol Roles

In every slot s, each party P executes the following roles:

IB Role:

(1) Check if eligible to produce an IB in slot s. If so, let π be the corresponding proof; otherwise, exit the procedure.
(2) Let txs be the (most recently stored) transactions. Set $\mathsf{IB} := (h, \text{txs})$ where $h := (s, P, \pi, x, \sigma)$, where x is the hash of b and σ is a signature of (the rest of) h under P's key in MI.
(3) Diffuse IB via F_{FFD}.

EB Role:

(1) Check if eligible to produce an EB in slot s. If so, let π be the corresponding proof; otherwise, exit the procedure.
(2) Link role: set L_{I} to be all (completely downloaded) IBs from $\text{slice}_L(s, \lambda + 1)$.
(3) Endorse role: set L_{E} to be all Vote1-certified EBs from $\text{slice}_L(s, \mu + 2)$.
(4) Diffuse $\mathsf{EB} := (s, P, \pi, L_{\mathsf{I}}, L_{\mathsf{E}}, \sigma)$ via F_{FFD}, where σ is a signature of (the rest of) EB under P's key in MI.

Vote-1 Role:

(1) Check if eligible to produce Vote1-vote in slot s.
(2) Create a vote for each EB from $\text{slice}_L(s, \mu + 1)$ for which all referenced IBs are fully downloaded.
(3) Diffuse the votes via F_{FFD}.

Vote-2 Role:

(1) Check if eligible to produce Vote2-vote in slot s.
(2) Create vote for each EB from $\text{slice}_L(s, 1)$ that references (i) a majority of all seen Vote1-certified EBs from $\text{slice}_L(s, \mu + 3)$ and (ii) no other EBs.
(3) Diffuse the votes via F_{FFD}.

Figure 2: The Simplified Leios Protocol.

3.3 The protocol

The Simplified Leios protocol is given in Figure 2. Initially, all parties locally generate keys for the IB/EB lottery as well as for voting, which they then register to the key registration functionality (cf. Section 2.2), that in turn distributes the public key list to all parties. Then, each party takes the voting public keys registered and uses them to compute a verification function $V_{\mathsf{chkCerts}}(\mathsf{EB}, c)$ that accepts if and only if c is a valid certificate for EB under the voting keys. This predicate is then passed to the base protocol, which replies with the stake distribution to be used in the protocol.

Simplified Leios then follows the intuition laid out above: in each slot, every party checks whether they are eligible to engage in any of the following roles:

- *IB role:* Creating IBs, corresponding to the Propose stage.
- *EB role:* Creating EBs; this role can be subdivided into a link role and an endorse role, corresponding to the Link and Endorse stages, respectively.
- *Vote-1 role:* Voting for EBs according to the Vote1 stage.
- *Vote-2 role:* Voting for EBs according to the Vote2 stage.

To illustrate this, assume $\lambda = \mu = 1$. In that case, there are seven active pipelines at any point, each of which is in a different stage. Assume they are numbered from 1 to 7 in ascending order of age (i.e., the newest one is pipeline 1).

Given this, a party plays the following roles in each pipeline:

- *Pipeline 1:* IB role;
- *Pipeline 2:* no role;
- *Pipeline 3:* link role;
- *Pipeline 4:* no role;
- *Pipeline 5:* vote-1 role;
- *Pipeline 6:* endorse role;
- *Pipeline 7:* vote-2 role.

Parties continuously input a most recent Vote2-certified endorser block to the base protocol, resulting in roughly every η/L-th pipeline being represented in the base chain. When there is no certified EB available (due to low participation—should the base protocol allow for dynamic participation), the parties simply input transactions received from the environment to the base protocol.

3.4 Analysis

The main part of the analysis is to show that every pipeline is "succesful," as defined formally below. Furthermore, the analysis bounds the liveness delay incurred by Simplified Leios. This section assumes that there is full participation. It is easy to see that under

low participation, Leios inherits safety and liveness from the base protocol.

3.4.1 Successful pipelines. A successful pipeline produces at least one Vote2-certified EB that covers all Propose-stage IBs:

Definition 3.1. A *pipeline instance* is *successful* if, by the end of the Vote2 stage, all honest input blocks from the Propose stage are (indirectly) referenced by a Vote2-certified EB from the Endorse stage.

Recall that the protocol execution is divided into slices of length L. In order for a slice to be "useful," the number of IBs generated during it must be in a reasonable interval, and an honest majority of the EBs generated must be from honest parties. This is captured by the following definition:

Definition 3.2. Let $\varepsilon > 0$. A *slice* is *ε-successful* if it produces

(1) at least $\alpha_{\mathrm{H}}(1-\varepsilon)Lf_{\mathrm{I}}$ honest IBs,[15]
(2) no more than $(1+\varepsilon)Lf_{\mathrm{I}}$ IBs, and
(3) an honest majority of endorser blocks.

The next step is to upper bound the probability of an unsuccessful slice. The proof of the following lemma applies Chernoff bounds and can be found in Appendix F.1.

Lemma 3.3. *Let ε such that $0 < \varepsilon < 2\alpha_{\mathrm{H}} - 1$. A slice is ε-successful except with probability $\varepsilon_{\mathrm{suc}}(L) := O\left(e^{-\varepsilon^2\Omega(f_{\mathrm{I}}L)}\right) + O\left(e^{-\varepsilon^2\Omega(f_{\mathrm{E}}L)}\right)$.*

Before proceeding, recall that F_{FFD} (cf. Section 2.1) guarantees delivery of an IB within $\tau(d+k)$ time where τ is the time to download the IB through a communication link, d is the diameter of the peer-to-peer network, and k is the number of IBs in transit.

In order to analyze the success of an entire pipeline instance, consider the following paramter settings (cf. Section 3.1 for intuition):

$$\rho := \frac{\tau d}{L}, \quad f_{\mathrm{I}} := \frac{\lambda-\rho}{(1+\varepsilon)(\lambda+1)} \cdot \frac{1}{\tau}, \quad \text{and} \quad \mu := \frac{\lambda^2-\rho}{1+\rho}.$$

Lemma 3.4. *For any $\varepsilon > 0$, a pipeline instance is successful if all its slices are ε-successful.*

Proof. It is sufficient to demonstrate the following facts:

(1) all honest IBs are delivered by the end of the Deliver1 stage; and
(2) all honest IBs seen by an honest party by the end of the Deliver1 stage are seen by all honest parties by the end of the Deliver2 stage.

Success of the Link stage then guarantees that all honest IBs from the Propose stage are referenced by all honest Vote1-certified EBs from the Link stage, and success of the Endorse stage guarantees that an (honest) Vote2-certified EB is produced that covers a majority of all Vote1-certified EBs of the Link stage. The latter implies that at least one honest Link-stage EB is referenced (as there is a majority of honest parties).

Fact (1) is proved by ensuring that the IB production during the first $\lambda+1$ pipeline slices essentially "fits" into the λL slots of the Deliver1 stage. Specifically, consider any honest IB from the Propose stage. As per the guarantees of F_{FFD}, the IB is globally delivered

$$\tau(d+k) \text{ slots after sending}$$

if at that point there are at most k IBs with newer slot numbers in the network.[16] Since all slices in the pipeline are assumed to be succesful, there are at most $(1+\varepsilon)(\lambda+1)Lf_{\mathrm{I}}$ IBs produced during all of Propose and Deliver1. Therefore, global delivery is guaranteed if

$$\tau(d+k) \;\le\; \tau\left(d + (1+\varepsilon)(\lambda+1)Lf_{\mathrm{I}}\right) \overset{!}{\le} \lambda L,$$

This is satisfied if

$$f_{\mathrm{I}} \;\le\; \frac{\lambda-\rho}{(1+\varepsilon)(\lambda+1)} \cdot \frac{1}{\tau},$$

which corresponds to the choice of f_{I} above.

One similarly argues that Fact (2) is true if the IB production during the first $\lambda+\mu+2$ slices fits into the $\mu+1$ slices corresponding to the Link and Deliver2 stages. This yields the condition

$$\tau\left(d + (1+\varepsilon)(\lambda+\mu+2)Lf_{\mathrm{I}}\right) \overset{!}{\le} (\mu+1)L,$$

which is satisfied if

$$f_{\mathrm{I}} \;\le\; \frac{(\mu+1)-\rho}{(1+\varepsilon)(\lambda+\mu+2)} \cdot \frac{1}{\tau}.$$

Given the choice of f_{I}, this is satisfied if μ is chosen such that

$$\frac{\lambda-\rho}{\lambda+1} \;\le\; \frac{(\mu+1)-\rho}{\lambda+\mu+2},$$

which requires

$$\mu \;\ge\; \frac{\lambda^2-\rho-1}{\rho+1}.$$

This is satisfied by the choice of μ above. □

Recall from Section 3.1 that the adversary can delay the availability of adversarial IBs referenced (indirectly) by Vote2-certified EBs. The following lemma quantifies this delay; the proof can be found in Appendix F.2.

Lemma 3.5. *Any IB (indirectly) referenced in a Vote2-certified EB from slot s is downloaded by every honest party no later than in slot $s + \nu L$ for*

$$\nu := \frac{(\lambda-\rho)(\lambda+\mu+2) + (\lambda+1)\rho}{\rho+1}.$$

Main theorems. In the following, let t be the time the protocol is run, $\varepsilon_{\mathrm{suc}}(L)$ the the probability a slice fails (cf. Lemma 3.3), and $\varepsilon_{\mathsf{vote}}(t)$ the error probability of the voting scheme if run for time t. Further, assume the base protocol satisfies persistence with parameter w and ledger quality η with liveness u; let $\varepsilon_{\mathsf{base}}(t)$ be the error of the base protocol when run for time t.

Theorem 3.6. *Let ε be such that $0 < \varepsilon < 2\alpha_{\mathrm{H}} - 1$. Then, Simplified Leios achieves persistence with parameter w' and liveness with parameters r' and u', where $w' = w$, $r' = \eta + (6+\lambda+\mu)L$, and $u' = u + (3+\lambda+\mu+\nu)L$, except with probability $O(t)\varepsilon_{\mathrm{suc}}(L) + \varepsilon_{\mathsf{vote}}(t) + \varepsilon_{\mathsf{base}}$.*

[15] Recall that α_{H} denotes the fraction of honest stake.

[16] Recall that τ is the time it takes to send a body over a single "hop" in the network.

Proof. The only way for persistence to be violated is if the adversary manages to forge an EB certificate for an EB that references IBs that are not available. Thus, under the assumption that there are no forgeries in the voting scheme, the extension protocol inherits persistence from the base protocol.

As for liveness, note that each slice is ε-successful except with probability $O(t)\varepsilon_{\mathsf{suc}}$, which follows from Lemma 3.3 and a union bound. To determine r', consider any interval of length η (the quality parameter of the base protocol), producing an honest base-protocol block (with some timestamp s). This block will (except for error $O(t)\varepsilon_{\mathsf{suc}}(L)$) reference a successful pipeline that started at time $s - d$ for $d \in [(6 + \lambda + \mu)L, (5 + \lambda + \mu)L]$, accounting for one pipeline length plus the fact that the base block may be desynchronized with the pipelines by up to one slice length L. As a consequence, any time interval $[t_0, t_0 + \eta]$ will produce a base-protocol block that references an honest IB from interval $[t_0 - (6+\lambda+\mu)L, t_1]$. Therefore, a transaction tx will be referenced in a base protocol block, if the honest parties receive it via SUBMIT during the r' slots.

As for u', any IB or transaction to be settled takes the extra detour via the pipeline (of duration $(5 + \lambda + \mu)L$) plus possible synchronization with the pipelines of duration L. Such an IB is thus represented in the base chain after time $(6 + \lambda + \mu)L + u$—although possibly not yet downloaded by value by all honest parties. By Lemma 3.5, those IB will be fully available νL after the Deliver2 stage, yielding the given u'. □

Theorem 3.7. *On average, Simplified Leios achieves throughput*

$$\frac{1}{1+\varepsilon} \cdot \frac{\lambda - \rho}{\lambda + 1} \cdot \alpha_{\mathrm{H}} \cdot \min(L/\eta, 1)$$

of honest input-block data relative to C, except with probability $O(t)\varepsilon_{\mathsf{suc}}(L) + \varepsilon_{\mathsf{vote}}(t) + \varepsilon_{\mathsf{base}}(t)$.

Recall that due to their small size compared to IBs, EBs and votes are modeled as consisting of headers only when it comes to diffusing them via F_{FFD}, and that the bandwidth needed for the base protocol is modeled separately (as hybrid functionalities assumed by the base protocol). In practice, this means that some of the overall bandwidth $\tilde{C}$ needs to be allocated to the base protocol and to the diffusion of EBs and votes. However, observe that the larger $\tilde{C}$ is, the smaller the fraction of bandwidth taken up by those parts becomes as the amount of traffic produced by them does not increase as one cranks up the IB data rate (by increasing the size of IBs).

Specifically, let ℓ be the bandwidth needed for the base protocol as well as the EBs and votes, i.e., $C = \tilde{C} - \ell$. Then, the throughput achieved by simplified Leios is (approximately)

$$\left(1 - \frac{1}{\lambda + 1}\right) \alpha_{\mathrm{H}} \eta' C \,,$$

where $\eta' := \min(L/\eta, 1)$. This can be written as $(1 - \delta')\alpha_{\mathrm{H}}\eta'\tilde{C}$ for

$$\delta' \;=\; \frac{1}{\lambda + 1} + \frac{O(\ell)}{\tilde{C}} \,.$$

Hence, for large enough total capacity $\tilde{C}$, Simplified Leios has a relative throughput arbitrarily close to

$$\left(1 - \frac{1}{\lambda + 1}\right) \alpha_{\mathrm{H}} \eta' \,.$$

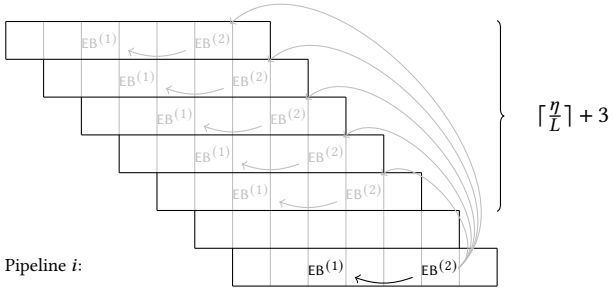

Figure 3: Augmented endorsing pipeline. A new pipeline starts every L slots, and from the roughly η/L previous pipelines, all succesful ones must be covered to guarantee full $\mathsf{EB}^{(2)}$ coverage despite of poor quality η of Π_{base}.

Finally, by choosing λ sufficiently large, one can get relative throughput $(1 - \delta)\alpha_{\mathrm{H}}\eta'$ for *any* $\delta > 0$.

Proof (of Theorem 3.7). Except with probability $O(t)\varepsilon_{\mathsf{suc}}(L)$, all slices are succesful, and thus, on average, the honest input-block data rate amounts to at least

$$f_{\mathrm{I}}|\mathsf{IB}| \;=\; \frac{1}{1+\varepsilon}\frac{\lambda - \rho}{\lambda + 1}\alpha_{\mathrm{H}}\frac{|\mathsf{IB}|}{\tau} \;=\; \frac{1}{1+\varepsilon}\frac{\lambda - \rho}{\lambda + 1}\alpha_{\mathrm{H}} C \,,$$

where the first equality follows from the choice of f_{I} and the second one is due to the fact that $\tau = |\mathsf{IB}|/C$.

Observe that a Vote2-certified EB from at least one out of every $\min(\eta/L, 1)$ pipelines is included in the base ledger by the chain quality assumption. The theorem follows. □

4 FULL LEIOS

The objective of Full Leios is to maintain high throughput independent of the ledger quality η of the base protocol. This is accomplished by modifying Simplified Leios (cf. Section 3) such that pipelines reference each other for a limited time horizon proportional to η.

To achieve this, first, the list $L_{\mathsf{E}} = (L_{\mathsf{E},1}, L_{\mathsf{E},2})$ is adapted so that it consists of two sublists $L_{\mathsf{E},1}$ and $L_{\mathsf{E},2}$, where $L_{\mathsf{E},1}$ takes on the role of L_{E} in Simplified Leios, and $L_{\mathsf{E},2}$ references EBs from previous pipelines. Concretely, the rules for the Endorse stage is adapted as follows (cf. Figure 3), where i is the index of the pipeline instance:

(6) Endorse: endorser blocks from this stage reference
   (a) in $L_{\mathsf{E},1}$, all Vote1-certified EB from the Link stage (as before), and
   (b) in $L_{\mathsf{E},2}$, for each the pipeline instances $i - \lceil \eta/L \rceil - 3, \ldots, i - 2$, one Vote2-certified EB from its Endorse stage—if such an EB exists.

The motivation for this rule change is that in order to guarantee eventual input-block inclusion in the base protocol by chain quality, a base-protocol block must be able to cover many pipeline instances—by storing one single certificate (to keep the block size minimal).

Base-protocol block inclusion and ledger extension. In each slot, parties input to the base ledger, via command SUBMIT, the most recent Vote2-certified EB (applying some tie-breaking) that satisfies the following properties (where i is the index of said EB candidate):

- for each index $j \in \{i - \lceil \eta/L \rceil - 3, \ldots, i-2\}$, if a Vote2-certified EB_j of that pipeline instance was locally seen Δ_{hdr} before the last slot of pipeline j, then EB must reference, in $L_{\mathsf{E},2}$, a Vote2-certified EB'_j of the same pipeline instance.

4.1 Analysis

It is easy to see that the guarantees of Simple Leios remain intact for Full Leios. We now state the additional guarantees, with respective proofs given in Appendix F.3.

Lemma 4.1. *Given a sequence of adjacent pipelines, the expected fraction of unsuccessful pipelines is at most* $\varphi_{\mathrm{suc}} = O(\lambda^2)\varepsilon_{\mathrm{suc}}(L)$.

In particular, note that, this rate vanishes exponentially fast in L.

Lemma 4.2. *Half of all successful pipelines are covered from the base protocol.*

Theorem 4.3. *Let* ε *be such that* $0 < \varepsilon < 2\alpha_{\mathrm{H}} - 1$. *Then, Full Leios achieves persistence with parameter* $w' = w$, *and, on average (over all transactions), the protocol achieves liveness with parameters* $r' = u'$, *and* $u' = u + \frac{L}{\varepsilon_{\mathrm{suc}}(L)} + O(\lambda^3)L$.

Note that if one were trying to hunt for the constant, one could also make *all* succesful pipelines referenced (possibly indirectly) from the base ledger, at the cost of doubling the size of the base-ledger payload (cf. proof of Lemma 4.2).

Theorem 4.4. *On average, Full Leios achieves a throughput of*

$$\frac{1}{1+\varepsilon} \cdot \frac{\lambda - \rho}{\lambda + 1} \cdot (1 - 2\varphi_{\mathrm{suc}}) \cdot \alpha_{\mathrm{H}}$$

of honest input-block data relative to C.

An argument similar to that after Theorem 3.7 shows that for large enough total capacity $\tilde{C}$, Full Leios has a relative throughput arbitrarily close to

$$\left(1 - \frac{1}{\lambda + 1}(1 - 2\varphi_{\mathrm{suc}})\right) \alpha_{\mathrm{H}} \,.$$

Finally, by choosing λ and L sufficiently large, one can get relative throughput $(1-\delta)\alpha_{\mathrm{H}}$ for *any* $\delta > 0$.

5 DEALING WITH EQUIVOCATIONS

Since block and vote production are controlled by a VRF-based lottery, the attacker can produce multiple valid blocks/votes per block/vote opportunity, known as "equivocations" and distribute different versions to different parties in the network. Consequently, the network must be equivocation-aware to avoid falling victim to such adversarial spamming. Furthermore, when the goal is to operate high-throughput protocols, an anti-equivocation strategy must ensure that among all equivocations for a particular block/vote opportunity, only one is fully downloaded (i.e., both header and body), as otherwise *half* of the available capacity is in jeopardy of being directly wasted. Conversely, downloading a *bounded* number of headers is acceptable since they are small compared to bodies.

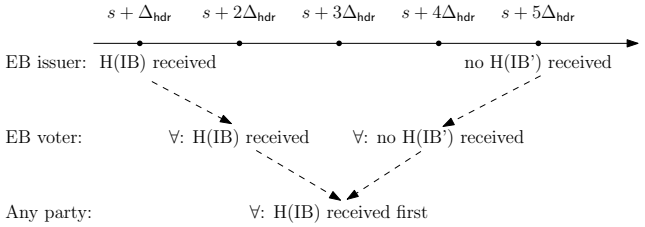

Figure 4: Timing rules for input blocks. $H(\mathsf{IB})$ **stands for "header of** IB**."**

This section presents a mechanism that guarantees that for each block/vote opportunity in the Leios extension protocol,

- each honest party downloads and forwards *at most two block headers*,
- each honest party downloads *at most one block fully*, and
- only the fully downloaded block/vote is required for the protocol (and, in particular, ledger correctness).

IB equivocations. Observe that for each block opportunity, F_{FFD} delivers to a party P at most one body, namely the body belonging to the first header P received by P;[17] below, the actual block downloaded by P is called the *preferred block for* P. It is therefore imperative to ensure the following for every IB opportunity:

- If the opportunity belongs to an honest party, the corresponding block produced must eventually be referenced from a Vote2-endorsed EB.
- If the opportunity belongs to a corrupted party, at most one corresponding IB may ever be referenced from a Vote2-endorsed EB, and the header for that IB must have reached all honest parties first.

To ensure this, parties apply the *timing rules* below (in addition to the normal protocol rules governing EB and vote production). Concretely, let $\mathsf{IB} = (h, b)$ be an IB from some slot s. Then,

(1) an EB producer P in the Link-phase only references IB in list L_{I} if P received (a) h before time $s + \Delta_{\mathsf{hdr}}$, and (b) no equivocating header h' before time $s + 5\Delta_{\mathsf{hdr}}$, and
(2) a voter P in the Vote1-phase only votes for an EB referencing IB if P received (a) h before time $s + 2\Delta_{\mathsf{hdr}}$ and (b) no equivocating header h' before time $s + 4\Delta_{\mathsf{hdr}}$.

These rules are illustrated in Figure 4. The intuition behind them is as follows: A voter must ensure that if it casts a Vote1-vote for an EB, then all IBs referenced therein will be the preferred IB for every honest party. Observe that if the voter does vote then, according to the protocol rules, they must have seen the headers of all referenced IBs (even their bodies) by $s + 2\Delta_{\mathsf{hdr}}$ but no headers of equivocations until $s + 4\Delta_{\mathsf{hdr}}$. Hence, by the guarantees of header delivery, all honest parties will receive the headers for the IBs referenced in the EB before the headers of any potential equivocations and therefore prefer those IBs.

In a similar spirit, an EB producer must ensure that the voters will vote for their EB. This adds an extra Δ_{hdr} on either side.

[17] As explained in Section 2.1 this behavior is straightforward to implement in the underlying gossip protocol.

EB equivocations. Dealing with EB equivocations proceeds along similar lines in order to ensure that an EB EB only becomes Vote2-certified if EB is the preferred block for the block opportunity corresponding to EB. Let EB = (h) be an EB from some slot s; recall that EBs consist of headers only. The timing rule is as follows: a voter P in the vote phase only votes for an EB if P received (a) h before time $s + \Delta_{\mathsf{hdr}}$ and (b) no equivocating header h' before time $s + 3\Delta_{\mathsf{hdr}}$.

Vote equivocations. The adversary may create multiple votes for the same vote opportunity. However, such an attack can be easily mitigated by having parties only download (and diffuse) the votes corresponding to the first EB header they have received for a given block opportunity.

Analysis. Based on the above, we can prove the following lemmas regarding anti-equivocation measures, which in turn can be easily used to extend the security theorems proved in Sections 3 and 4 to the setting with block-equivocations. We point to Appendix F.4 for the full proofs.

Lemma 5.1. *For each block opportunity, each honest party downloads at most two block headers and one block body.*

Lemma 5.2. *If* EB *is voted by an honest party, then all* IB*s referenced in* EB *are downloaded by all honest parties.*

Lemma 5.3. *If* EB *is voted by an honest party, then* EB *gets downloaded by all honest parties.*

Lemma 5.4. *Given that the respective pipeline is successful, every* EB *produced by an honest party gets voted for by each honest party.*

6 BLOCK VOTING

Block data availability and correctness in Leios is ensured by having parties vote on it: a block is deemed correct and available if a majority of the stake is voting for it. Moreover, voting is used to ensure that EBs containing honestly produced IBs are included in the main chain. This makes the voting scheme used in Leios a central piece of its architecture. We explore voting requirements and provide a practical instantiation along the lines of previous works [6, 7, 28] in Appendix D.

7 GENERALIZATIONS

Our protocol can also span multiple "epochs" (reflecting underlying stake change over the course of time) and is applicable to base protocols based on other resources than stake, e.g., proof of work (PoW). Details on this are given in Appendix E.

REFERENCES

[1] Zeta Avarikioti, Antoine Desjardins, Lefteris Kokoris-Kogias, and Roger Wattenhofer. 2023. Divide & Scale: Formalization and Roadmap to Robust Sharding. In *International Colloquium on Structural Information and Communication Complexity*. Springer, 199–245.

[2] Zeta Avarikioti, Lioba Heimbach, Roland Schmid, Laurent Vanbever, Roger Wattenhofer, and Patrick Wintermeyer. 2020. Fnf-BFT: Exploring performance limits of bft protocols.

[3] Vivek Kumar Bagaria, Sreeram Kannan, David Tse, Giulia C. Fanti, and Pramod Viswanath. 2019. Prism: Deconstructing the Blockchain to Approach Physical Limits. In *ACM CCS 2019*, Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz (Eds.). ACM Press, 585–602. https://doi.org/10.1145/3319535.3363213

[4] Mihir Bellare, Juan A. Garay, and Tal Rabin. 1998. Fast Batch Verification for Modular Exponentiation and Digital Signatures. In *EUROCRYPT'98 (LNCS, Vol. 1403)*, Kaisa Nyberg (Ed.). Springer, Heidelberg, 236–250. https://doi.org/10.1007/BFb0054130

[5] Joseph Bonneau, Andrew Miller, Jeremy Clark, Arvind Narayanan, Joshua A. Kroll, and Edward W. Felten. 2015. SoK: Research Perspectives and Challenges for Bitcoin and Cryptocurrencies. In *2015 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 104–121. https://doi.org/10.1109/SP.2015.14

[6] Pyrros Chaidos and Aggelos Kiayias. 2021. Mithril: Stake-based Threshold Multisignatures. Cryptology ePrint Archive, Report 2021/916. https://eprint.iacr.org/2021/916.

[7] Pyrros Chaidos, Aggelos Kiayias, Leonid Reyzin, and Anatoliy Zinovyev. 2023. Approximate Lower Bound Arguments. *Cryptology ePrint Archive* (2023).

[8] Manuel M. T. Chakravarty, James Chapman, Kenneth MacKenzie, Orestis Melkonian, Michael Peyton Jones, and Philip Wadler. 2020. The Extended UTXO Model. In *FC 2020 Workshops (LNCS, Vol. 12063)*, Matthew Bernhard, Andrea Bracciali, L. Jean Camp, Shin'ichiro Matsuo, Alana Maurushat, Peter B. Rønne, and Massimiliano Sala (Eds.). Springer, Heidelberg, 525–539. https://doi.org/10.1007/978-3-030-54455-3_37

[9] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed E. Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, Dawn Song, and Roger Wattenhofer. 2016. On Scaling Decentralized Blockchains - (A Position Paper). In *FC 2016 Workshops (LNCS, Vol. 9604)*, Jeremy Clark, Sarah Meiklejohn, Peter Y. A. Ryan, Dan S. Wallach, Michael Brenner, and Kurt Rohloff (Eds.). Springer, Heidelberg, 106–125. https://doi.org/10.1007/978-3-662-53357-4_8

[10] George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. 2022. Narwhal and tusk: a dag-based mempool and efficient bft consensus. In *Proceedings of the Seventeenth European Conference on Computer Systems*. 34–50.

[11] George Danezis and Sarah Meiklejohn. 2016. Centrally Banked Cryptocurrencies. In *NDSS 2016*. The Internet Society.

[12] Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. 2018. Ouroboros Praos: An Adaptively-Secure, Semi-synchronous Proof-of-Stake Blockchain. In *EUROCRYPT 2018, Part II (LNCS, Vol. 10821)*, Jesper Buus Nielsen and Vincent Rijmen (Eds.). Springer, Heidelberg, 66–98. https://doi.org/10.1007/978-3-319-78375-8_3

[13] Amir Dembo, Sreeram Kannan, Ertem Nusret Tas, David Tse, Pramod Viswanath, Xuechao Wang, and Ofer Zeitouni. 2020. Everything is a Race and Nakamoto Always Wins. In *ACM CCS 2020*, Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna (Eds.). ACM Press, 859–878. https://doi.org/10.1145/3372297.3417290

[14] Thomas D. Dickerson, Paul Gazzillo, Maurice Herlihy, and Eric Koskinen. 2017. Adding Concurrency to Smart Contracts. In *36th ACM PODC*, Elad Michael Schiller and Alexander A. Schwarzmann (Eds.). ACM, 303–312. https://doi.org/10.1145/3087801.3087835

[15] Devdatt P Dubhashi and Alessandro Panconesi. 2009. *Concentration of measure for the analysis of randomized algorithms*. Cambridge University Press.

[16] Ittay Eyal and Emin Gün Sirer. 2014. Majority Is Not Enough: Bitcoin Mining Is Vulnerable. In *FC 2014 (LNCS, Vol. 8437)*, Nicolas Christin and Reihaneh Safavi-Naini (Eds.). Springer, Heidelberg, 436–454. https://doi.org/10.1007/978-3-662-45472-5_28

[17] Matthias Fitzi, Peter Gaži, Aggelos Kiayias, and Alexander Russell. 2018. Parallel Chains: Improving Throughput and Latency of Blockchain Protocols via Parallel Composition. Cryptology ePrint Archive, Report 2018/1119. https://eprint.iacr.org/2018/1119.

[18] Matthias Fitzi and Martin Hirt. 2006. Optimally efficient multi-valued byzantine agreement. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Principles of Distributed Computing, PODC 2006, Denver, CO, USA, July 23-26, 2006*, Eric Ruppert and Dahlia Malkhi (Eds.). ACM, 163–168. https://doi.org/10.1145/1146381.1146407

[19] Matthias Fitzi, Xuechao Wang, Sreeram Kannan, Aggelos Kiayias, Nikos Leonardos, Pramod Viswanath, and Gerui Wang. 2022. Minotaur: Multi-resource blockchain consensus. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 1095–1108.

[20] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. 2015. The Bitcoin Backbone Protocol: Analysis and Applications. In *EUROCRYPT 2015, Part II (LNCS, Vol. 9057)*, Elisabeth Oswald and Marc Fischlin (Eds.). Springer, Heidelberg, 281–310. https://doi.org/10.1007/978-3-662-46803-6_10

[21] Peter Gazi, Aggelos Kiayias, and Alexander Russell. 2020. Tight Consistency Bounds for Bitcoin. In *ACM CCS 2020*, Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna (Eds.). ACM Press, 819–838. https://doi.org/10.1145/3372297.3423365

[22] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. 2017. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th symposium on operating systems principles*. 51–68.

[23] Suyash Gupta, Jelle Hellings, and Mohammad Sadoghi. 2019. RCC: Resilient Concurrent Consensus for High-Throughput Secure Transaction Processing. arXiv preprint arXiv:1911.00837.

[24] Harry A. Kalodner, Steven Goldfeder, Xiaoqi Chen, S. Matthew Weinberg, and Edward W. Felten. 2018. Arbitrum: Scalable, private smart contracts. In *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*, William Enck and Adrienne Porter Felt (Eds.). USENIX Association, 1353–1370. https://www.usenix.org/conference/usenixsecurity18/presentation/kalodner

[25] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. 2017. Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol. In *CRYPTO 2017, Part I (LNCS, Vol. 10401)*, Jonathan Katz and Hovav Shacham (Eds.). Springer, Heidelberg, 357–388. https://doi.org/10.1007/978-3-319-63688-7_12

[26] Lucianna Kiffer, Joachim Neu, Srivatsan Sridhar, Aviv Zohar, and David Tse. 2023. Security of Nakamoto Consensus under Congestion. Cryptology ePrint Archive, Paper 2023/381. https://eprint.iacr.org/2023/381 https://eprint.iacr.org/2023/381.

[27] Silvio Micali, Michael O. Rabin, and Salil P. Vadhan. 1999. Verifiable Random Functions. In *40th FOCS*. IEEE Computer Society Press, 120–130. https://doi.org/10.1109/SFFCS.1999.814584

[28] Silvio Micali, Leonid Reyzin, Georgios Vlachos, Riad S Wahby, and Nickolai Zeldovich. 2021. Compact certificates of collective knowledge. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 626–641.

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

[30] Kamilla Nazirkhanova, Joachim Neu, and David Tse. 2021. Information Dispersal with Provable Retrievability for Rollups. Cryptology ePrint Archive, Report 2021/1544. https://eprint.iacr.org/2021/1544.

[31] Joachim Neu, Srivatsan Sridhar, Lei Yang, David Tse, and Mohammad Alizadeh. 2022. Longest chain consensus under bandwidth constraint. In *Proceedings of the 4th ACM Conference on Advances in Financial Technologies*. 126–147.

[32] Rafael Pass, Lior Seeman, and abhi shelat. 2017. Analysis of the Blockchain Protocol in Asynchronous Networks. In *EUROCRYPT 2017, Part II (LNCS, Vol. 10211)*, Jean-Sébastien Coron and Jesper Buus Nielsen (Eds.). Springer, Heidelberg, 643–673. https://doi.org/10.1007/978-3-319-56614-6_22

[33] Rafael Pass and Elaine Shi. 2017. FruitChains: A Fair Blockchain. In *36th ACM PODC*, Elad Michael Schiller and Alexander A. Schwarzmann (Eds.). ACM, 315–324. https://doi.org/10.1145/3087801.3087809

[34] Alexander Spiegelman, Neil Giridharan, Alberto Sonnino, and Lefteris Kokoris-Kogias. 2022. Bullshark: Dag bft protocols made practical. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 2705–2718.

[35] Chrysoula Stathakopoulou, David Tudor, Matej Pavlovic, and M Vukolić. 2022. Mir-BFT: Scalable and Robust BFT for Decentralized Networks. *Journal of Systems Research* 2, 1 (2022).

[36] Haifeng Yu, Ivica Nikolic, Ruomu Hou, and Prateek Saxena. 2020. OHIE: Blockchain Scaling Made Simple. In *2020 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 90–105. https://doi.org/10.1109/SP40000.2020.00008

A NETWORKING FUNCTIONALITY

The networking functionality F_{FFD}, the intuition behind which is explained in Section 2.1, is depicted in Figure 5.

State. The functionality keeps track of all headers and bodies of diffused blocks in sets Hdrs and Bdys, respectively, along with the time t of initial diffusion and the set of parties $\mathcal{P}$ that have already received the header and body, respectively. For bodies, the functionality also records the corresponding header.

In addition, for each party P and message ID mid, the functionality records the preferred header of P for mid.

Diffusion. In order to diffuse a new block (h, b) (with $\mathsf{match}(h, b)$), the sender calls (DiffFB, h, b). There is also a diffusion command DiffHdr for diffusing headers only. Recall from Section 2.1 that validity checks are performed externally, i.e., by a party re-inputting valid headers via DiffHdr after having received them via FetchHdrs (see below).

Equivocations. Based on headers input by parties, the functionality identifies equivocated blocks. Specifically, two different headers for the same mid input via the diffusion commands by an honest party, and thus assumed valid, constitute a *proof of equivocation (PoE).*

Fetching. Functionality F_{FFD} offers two fetch-related commands FetchHdrs and FetchBdys for fetching headers and bodies, respectively. Headers are guaranteed to be delivered within Δ_{hdr} after initial diffusion, but the adversary may deliver them any time before that. Note that once a party has seen a PoE for a particular mid, the functionality no longer delivers any headers for mid.

As for fetching bodies, first note that F_{FFD} only ever delivers the bodies belonging to the preferred headers. Furthermore, for equivocated blocks, the body delivery guarantees are limited: they are only delivered provided that all honest parties have the same preferred header (cf. condition 2(a)(ii) in paragraph "Bodies" in Figure 5). This is due to the fact that in the motivating gossip protocol underlying F_{FFD}, parties only download bodies for their preferred headers, and therefore, global body delivery can only be guaranteed if all honest parties have the same preferred header for an ID mid.

Finally, the delivery delay is

$$\tau(d + k) \text{ time after initial diffusion}$$

(or after the first honest party has seen it) if at that point there are at most k messages with newer message IDs in the network, where $\tau := |b|/C$ is the time it takes to send a body b over a single "hop" in the network. (For simplicity, it is assumed that all bodies have the same size.) The rationale behind this is as follows: if there are no fresher messages impeding the delivery of a particular message m, then m reaches all parties after being in transit for at most τd time, while each newer message causes m to be queued at most once. Note that messages for which there exist double signs enjoy no such guarantees.

Again, the adversary may cause a message to be delivered at any time before the above delay.

B VERIFIABLE RANDOM FUNCTIONS

A *verifiable random function (VRF)*, defined by Micali *et al.* [27], is a cryptographic primitive that allows a party P to create a key pair consisting of a secret evaluation key and a public verification key such that: (a) the public key implicitly defines a function f (b) with the secret key, P can evaluate f at any input x, obtaining an output y and a proof π; (c) given the public key of P, anyone is able to verify, using π, that y is indeed the output corresponding to x; (d) with only the public verification key, outputs of the function cannot be predicted, and in fact appear random.

The above guarantees are abstracted by an idealized functionality F_{vrf} (cf. Figure 6). The main commands offered by F_{vrf} are (i) (GetKey), answered by (GetKey, v) for an (adversarially chosen) idealized public key v, (ii) (Eval, v, x), answered by (Eval, y, π), and (iii) (Verify, v, x, y, π), answered by (Verify, ϕ) with $\phi \in \{0, 1\}$ indicating whether π is a valid proof for the input/output pair (x, y) under public key v.

C CHERNOFF-HOEFFDING BOUNDS

We make use of the Chernoff-Hoeffding bound in form of Eq. (1.7) in [15].

Theorem C.1 ([15]). *Let $X := \sum_{i\in[n]} X_i$ where X_i, $i \in [n]$ are independently distributed in $[0, 1]$. Then, for $0 < \varepsilon < 1$,*

$$\Pr[X > (1 + \varepsilon)E[X]] \leq \exp\left(-\frac{\varepsilon^2}{3}E[X]\right),$$

and

$$\Pr[X < (1 - \varepsilon)E[X]] \leq \exp\left(-\frac{\varepsilon^2}{2}E[X]\right),$$

D BLOCK VOTING

Block data availability and correctness in Leios is ensured by having parties vote on it: a block is deemed correct and available if a majority of the stake is voting for it. Moreover, voting is used to ensure that EBs containing honestly produced IBs are included in the main chain. This makes the voting scheme used in Leios a central piece of its architecture. In this section, we explore voting requirements and provide as a proof-of-concept a practical instantiation along the lines of previous works [6, 7, 28].

D.1 Requirements

Voting in Leios must meet two key requirements: Firstly, a voting certificate must fit in a base-protocol blockto ensure agreement on the serialization of IBs. At the same time certificate generation must be as fast as possible, as it is part of preparing the input for the base-protocol which may be a time-sensitive operation, e.g, delaying the release of a block in Bitcoin by 5 minutes has a negative effect on the overall security of the protocol. Finally, the communication complexity of the voting protocol should be kept at minimum, to maintain a high-throughput to capacity rate. To address this situation, we describe next a simple voting scheme based on BLS signatures and argue that it can be securely integrated with our main protocol.

D.2 Protocol description

On a high level, our scheme consists of the following elements: A VRF-based voting eligibility mechanism (similarly to block production) to ensure that the number of votes depends only weakly to

Functionality F_{FFD}

Parameters & Notation: $\Delta_{\mathsf{hdr}} \in \mathbb{N}$, header propagation delay; $d \in \mathbb{N}$, diameter; C, capacity of the network links; $\mathsf{match}(h, b)$, true iff body b matches header h; $\mathsf{msgID}(h)$, message ID of h. $\mathbb{T}$ denotes the current time.

Admin

Variables: The functionality keeps track of the following variables, initialized to the values below:

- $\mathsf{Hdrs} := \emptyset$: contains a triple $(h, t, \mathcal{P})$ for each header h diffused, where t is the time of diffusion and $\mathcal{P}$ the set of parties who have already received it.
- $\mathsf{prefHdr}_P[\mathrm{mid}] := \perp$: preferred header of P for message with ID mid.
- $\mathsf{Bdys} := \emptyset$: contains a tuple $(h, b, t, \mathcal{P})$ for header h and b diffused, where t is the time of diffusion and $\mathcal{P}$ the set of parties who have already received it.

Helpers: The description uses the following helpers:

- $\mathsf{hasHdr}(P, \mathrm{mid})$: returns true iff there exists $(h, \cdot, \mathcal{P}) \in \mathsf{Hdrs}$ with $\mathsf{msgID}(h) = \mathrm{mid}$ and $P \in \mathcal{P}$.
- $\mathsf{hasPoE}(P, \mathrm{mid})$: returns true iff there exist $(h, \cdot, \mathcal{P}), (h', \cdot, \mathcal{P}') \in \mathsf{Hdrs}$ with $\mathsf{msgID}(h) = \mathsf{msgID}(h') = \mathrm{mid}$ and $P \in \mathcal{P} \cap \mathcal{P}'$.
- $\mathsf{hasBdy}(P, \mathrm{mid})$: returns true iff there exists $(h, \cdot, \cdot, \mathcal{P}) \in \mathsf{Bdys}$ with $\mathsf{msgID}(h) = \mathrm{mid}$ and $P \in \mathcal{P}$.
- $\mathsf{HdrsAdd}(h, t, P)$: if there exists $(h, \cdot, \mathcal{P}) \in \mathsf{Hdrs}$, add P to $\mathcal{P}$; otherwise, add $(h, t, \{P\})$ to Hdrs.
- $\mathsf{BdysAdd}(h, b, t, P)$: if there exists $(h, \cdot, \cdot, \mathcal{P}) \in \mathsf{Hdrs}$, add P to $\mathcal{P}$; otherwise, add $(h, b, t, \{P\})$ to Bdys.
- $\mathsf{newerBdys}(h)$: returns the number mid for which there exists $(h', \cdot, \cdot, \cdot) \in \mathsf{Bdys}$ with $\mathsf{msgID}(h') > \mathsf{msgID}(h)$.

Diffusion

Full block: Upon $(\textsc{DiffFB}, h, b)$ from honest P:

(1) Require: $\mathsf{match}(h, b)$ and $\neg\mathsf{hasHdr}(P, \mathsf{msgID}(h))$.

(2) Bookkeeping: $\mathsf{HdrsAdd}(h, \mathbb{T}, P)$, $\mathsf{prefHdr}_P[\mathsf{msgID}(h)] := h$, and $\mathsf{BdysAdd}(h, b, \mathbb{T}, P)$.

Header: Upon $(\textsc{DiffHdr}, h)$ from honest P:

(1) Require: $\neg\mathsf{hasPoE}(P, \mathsf{msgID}(h))$.

(2) Bookkeeping: $\mathsf{HdrsAdd}(h, \mathbb{T}, P)$, and if $\mathsf{prefHdr}_P(\mathsf{msgID}(h)) = \perp$, set it to h.

Fetching

Headers: Upon $(\textsc{FetchHdrs})$ from honest P:

(1) Initialize "to-be-delivered" set $D \leftarrow \emptyset$.

(2) For all $(h, t, \mathcal{P}) \in \mathsf{Hdrs}$ within $P \notin \mathcal{P}$:
  (a) If $\mathbb{T} \geq t + \Delta_{\mathsf{hdr}}$ and $\neg\mathsf{hasPoE}(P, \mathsf{msgID}(h))$, add h to D.

(3) Output $(\textsc{FetchHdrs}, P)$ to $\mathcal{A}$ and get a set $D_{\mathcal{A}}$ from $\mathcal{A}$.

(4) Remove from $D_{\mathcal{A}}$ all h with $\mathsf{hasPoE}(P, \mathsf{msgID}(h))$.

(5) Output $(\textsc{FetchHdrs}, D \cup D_{\mathcal{A}})$ to P.

Bodies: Upon $(\textsc{FetchBdys})$ from honest P:

(1) Initialize "to-be-delivered" set $D \leftarrow \emptyset$.

(2) For all $(h, b, t, \mathcal{P}) \in \mathsf{Bdys}$ with $P \notin \mathcal{P}$:
  (a) Add (h, b) to D if the following conditions are met:
    (i) $\mathsf{prefHdr}_P[\mathsf{msgID}(h)] = h$;
    (ii) for all honest P': $\mathsf{prefHdr}_{P'}[\mathsf{msgID}(h)] \in \{h, \perp\}$;
    (iii) there exists $k \geq 0$ s.t. (a) $\mathbb{T} \geq t + (d + k)\,|b|/C$ and (b) $\mathsf{newerBdys}(h) \leq k$.

(3) Output $(\textsc{FetchBdys}, P)$ to $\mathcal{A}$ and get a set $D_{\mathcal{A}}$ from $\mathcal{A}$.

(4) For each $(h, b) \in D_{\mathcal{A}}$, if (a) $\mathsf{match}(h, b)$, (b) $\neg\mathsf{hasBdy}(P, \mathsf{msgID}(h))$, and (c) $\mathsf{prefHdr}_P(\mathsf{msgID}(h)) = h$: add (h, b) to D.

(5) For each $(h, b) \in D$, $\mathsf{BdysAdd}(h, b, \mathbb{T}, P)$.

(6) Output $(\textsc{FetchBdys}, D)$ to P.

Figure 5: Functionality F_{FFD}

the total number of parties, which for blockchain protocols can be rather large. A voting procedure that allows parties to create a vote for the choice they support, i.e, for our use case a candidate EB that satisfies the properties outlined in Sections 3 and 4. Finally, a certificate generation mechanism that allows parties to create a compact certificate, when provided with a sufficient amount of same-preference votes. Importantly, a certificate can only be created if a set of parties controlling a majority of the stake supports it.

The syntax of our scheme is presented next, with the detailed code shown in Figures 7 and 8.

- Algorithm KeyGen(Param), takes as input the scheme parameters, and outputs a pair of cryptographic keys (sk, mvk) and a proof of possession of the secret key μ.
- Algorithm GenVote(eid, m, sk, mvk, s), takes as input an election id eid, a message m, a pair of keys (sk, mvk), and the amount of stake s associated with these keys, and outputs either $\perp$ or a vote v. [18]
- Algorithm VerifyVote(eid, m, mvk, s, v), takes as input an election id eid, a message m, a verification key mvk, the associated stake amount s, and a vote v, and outputs true or false.
- Algorithm GenCert$(eid, m, \boldsymbol{mvk}, \boldsymbol{s}, \boldsymbol{v})$, takes as input an election id eid, a message m, and lists of verification keys, stake amounts, and votes, and outputs a vote certificate c or $\perp$.
- Algorithm VerifyCert(eid, m, c) takes as input an election id eid, a message m, and a certificate c, and outputs true or false.

Parties generate their voting keys by running the KeyGen(Param) procedure. The keys are basically a pair of BLS keys together with a proof of possession over the pairing-friendly elliptic curve defined in Param. Parties are expected to register their keys on the blockchain ahead of time to participate in the voting. [19]

[18] As we will see later, for efficiency reasons not all parties may be eligible to vote. In this case, we expect the algorithm to output $\perp$.

[19] In the case of PoS blockchains, key registration is anyway a standard operation.

Functionality F_{vrf}

Variables: The functionality keeps track of the following variables, initialized to the values below:

(1) an array $\text{Keys}[P] := \emptyset$: set of keys owned by P;
(2) an array $T[v, x] := \perp$, where v is a key and x a domain value: pair (y, S), where y is a range value and S a set of proofs π;
(3) a set $E := \emptyset$: contains triples (v, x, y) to keep track of all VRF evaluations.

Keys: Upon (GETKEY) from P: Output (GETKEY, P) to $\mathcal{S}$ and ask $\mathcal{S}$ for a unique key v. Add v to $\text{Keys}[P]$ and output (GETKEY, v) to P.

Register Keys: Upon (REGKEY, v) from $\mathcal{S}$: if v is unique, add it to $\text{Keys}[\mathcal{S}]$.

Evaluation: Upon (EVAL, v, x) from P with $v \in \text{Keys}[P]$: Output (EVAL, P, v, x) to $\mathcal{S}$, and upon obtaining (EVAL, π) from $\mathcal{S}$:

(1) If π is not unique, exit the procedure.
(2) If $T[v, x]$ is undefined, pick y uniformly at random from the range and set $S := \emptyset$; otherwise, let $(y, S) := T[v, x]$.
(3) Set $T[v, x] := (y, S \cup \{\pi\})$ and add (v, x, y) to E.
(4) Output (EVAL, y, π) to P.

Malicious evaluation: Upon (EVAL, v, x) from $\mathcal{S}$:

- *Case 1:* There exists an *uncorrupted* P with $v \in \text{Keys}[P]$: If $(y, S) := T[v, x]$ is defined, return (EVAL, y) to $\mathcal{S}$; otherwise, do nothing.
- *Case 2:* There exists a *corrupted* P with $v \in \text{Keys}[P]$ *or* $v \in \text{Keys}[\mathcal{S}]$: If $T[v, x]$ is not defined, pick y uniformly at random from the range, let $S := \emptyset$ and set $T[v, x] := (y, S)$; otherwise, let $(y, S) := T[v, x]$. Return (EVAL, y) to $\mathcal{S}$.
- *Else:* Do nothing.

Verification: Upon (VERIFY, v, x, y, π) from any ITI: Send (VERIFY, v, x, y, π) to $\mathcal{S}$, and upon receiving (VERIFY, ϕ) from $\mathcal{S}$:

- If $v \in \text{Keys}[\cdot]$ and $T[v, x] = (y, S)$ is defined:
  (1) If $\pi \in S$, set $f := 1$.
  (2) Else, if $\phi = 1$ and π is unique—i.e., if for all $(v', x') \neq (v, x)$ with $T[v', x'] = (\cdot, S)$, $\pi \notin S$—set $T[v, x'] := (y, S \cup \{\pi\})$ and $f := 1$.
  (3) Else, set $f := 0$.
- Else, set $f := 0$.

Output (VERIFY, f) to P.

Adversarial leakage: Upon (LEAK) from $\mathcal{S}$: return (LEAK, E) to $\mathcal{S}$.

Figure 6: VRF functionality F_{vrf}.

Algorithm BLS

Parameters & Notation: E is a pairing-elliptic curve E on $\mathbb{F}_p$, forming groups $\mathbb{G}_1, \mathbb{G}_2$ of order q, with pairing function $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. g_1, g_2 are generators of $\mathbb{G}_1, \mathbb{G}_2$, respectively. We use $\text{BLS.Setup}(1^\kappa)$ to refer to the function that generates the group parameters $\text{Param} := (\mathbb{G}_1, \mathbb{G}_2, g_1, g_2, q, e, \mathbb{G}_T, p)$ for BLS. We make use of hash functions $H_{\mathbb{G}_1} : \{0,1\}^* \to \mathbb{G}_1$, $H_q : \{0,1\}^* \to \mathbb{Z}_q$, $H_p : \{0,1\}^* \to \mathbb{F}_p$, modeled as random oracles in the corresponding groups. For batching we also use a truncated version of H_q, $H_\kappa : \{0,1\}^* \to \mathbb{Z}_{2^\kappa}$.

BLS.KeyGen(Param):

(1) $sk \leftarrow \mathbb{Z}_q$
(2) $mvk := g_2^{sk}$
(3) $\mu_1 := H_{\mathbb{G}_1}(\text{“}PoP\text{”}||mvk)^{sk}$
(4) $\mu_2 := g_1^{sk}$
(5) return $(sk, mvk, \boldsymbol{\mu} := (\mu_1, \mu_2))$

BLS.Check$(mvk, \boldsymbol{\mu})$**:**

(1) $b_1 := (e(\mu_1, g_2) \stackrel{?}{=} e(H_{\mathbb{G}_1}(\text{“}PoP\text{”}||mvk), mvk))$
(2) $b_2 := (e(g_1, mvk) \stackrel{?}{=} e(\mu_2, g_2))$
(3) return $b_1 \wedge b_2$

BLS.Sign(sk, m)**:**

(1) return $H_{\mathbb{G}_1}(\text{“}M\text{”}||m)^{sk}$

BLS.Ver(mvk, m, σ)**:**

(1) return $e(\sigma, g_2) \stackrel{?}{=} e(H_{\mathbb{G}_1}(\text{“}M\text{”}||m), mvk)$

BLS.AKey$(\boldsymbol{mvk})$**:**

Keys in $\boldsymbol{mvk}$ are assumed to be previously checked.

(1) $ivk := \prod_{mvk_i \in \boldsymbol{mvk}}(mvk_i)$
(2) return ivk

BLS.ASig$(\boldsymbol{\sigma})$**:**

(1) $\tilde{\sigma} := \prod_{\sigma_i \in \boldsymbol{\sigma}}(\sigma_i)$
(2) return $\tilde{\sigma}$

BLS.BKey$(\boldsymbol{mvk}, \boldsymbol{\sigma})$**:**

Keys in $\boldsymbol{mvk}$ are assumed to be previously checked.

(1) $e_\sigma := H_p(\boldsymbol{\sigma})$
(2) $ivk := \prod_{mvk_i \in \boldsymbol{mvk}} mvk_i^{H_\mu(i, e_\sigma)}$
(3) return ivk

BLS.BSig$(\boldsymbol{\sigma})$**:**

(1) $e_\sigma := H_p(\boldsymbol{\sigma})$
(2) $\tilde{\sigma} := \prod_{\sigma_i \in \boldsymbol{\sigma}} \sigma_i^{H_\mu(i, e_\sigma)}$
(3) return $\tilde{\sigma}$

Figure 7: The modified BLS scheme.

Next, whenever an election is initiated in the protocol, i.e, an EB is produced, and assuming the necessary conditions for supporting an EB are met, parties attempt to generate a vote using procedure GenVote. The election id of the vote corresponds to the VRF proof of the EB, while the message corresponds to the header of the EB. GenVote first checks whether the party is eligible to create a vote by initially BLS-signing the election id and then doing a number (m_v) of hash-and-compare checks (helper procedure VoteCount in Figure 8). If any of these checks succeeds, the party is eligible to produce a vote by computing an additional BLS-signature on the

Algorithm Π_{vote}

Parameters & Notation:
We make use of a hash function $H : \{0,1\}^* \to \{0,1\}^\kappa$ modeled as a random oracle. The Proof-of-Possession of voting keys are assumed to be verified when the keys are registered. Map $T : [0,1] \to \{0,\ldots,2^\kappa\}$ is set such that an eligibility check succeeds with probability $\phi(s) := 1-(1-f)^s$, for parameter f. Parameters m_v, k_v denote the number of eligibility attempts and the minimum number of votes required in a certificate, respectively.

Helper Procedures

VoteCount(x, s)**:**
(1) $y := H(x)$
(2) $cnt := \sum_{i\in[m_v]}(H(y,i) > T(s))$
(3) return cnt;

Voting Procedures

KeyGen$(\mathbf{Param})$**:**
(1) $(sk, mvk, \boldsymbol{\mu}) :=$ BLS.KeyGen(Param)
(2) return $(sk, mvk, \boldsymbol{\mu})$

GenVote(eid, m, sk, s)**:**
(1) $\sigma_{eid} :=$ BLS.Sign(sk, eid)
(2) if (VoteCount$(\sigma_{eid}, s) < 1$) return $\perp$
(3) $\sigma_m :=$ BLS.Sign$(sk, eid||m)$
(4) return (σ_{eid}, σ_m)

VerifyVote$(eid, m, mvk, s, (\sigma_{eid}, \sigma_m))$**:**
(1) if (VoteCount$(\sigma_{eid}, s) < 1$) return $\perp$
(2) return (BLS.Ver$(mvk, eid, \sigma_{eid}) \wedge$ BLS.Ver$(mvk, eid||m, \sigma_m)$)

GenCert$(eid, m, \boldsymbol{mvk}, s, \boldsymbol{v})$ **:**
Assuming individual votes are valid.
(1) $((\sigma_i, \sigma'_i))_i := \boldsymbol{v}$; $(s_i)_i := s$
(2) If $(\sum_i$ VoteCount$(\sigma_i, s_i) < k_v)$ return $\perp$
(3) $\tilde{\sigma}_{eid} :=$ BLS.BSig$(eid, (\sigma_i)_i)$
(4) $\tilde{\sigma}_m :=$ BLS.ASig$(eid||m, (\sigma'_i)_i)$
(5) return $((mvk_i, s_i, \sigma_i)_i, \tilde{\sigma}_{eid}, \tilde{\sigma}_m)$

VerifyCert(eid, m, c) **:**
(1) $((mvk_i, s_i, \sigma_i)_i, \tilde{\sigma}_{eid}, \tilde{\sigma}_m) := c$
(2) If $(\sum_i$ VoteCount$(\sigma_i, s_i) < k_v)$ return $\perp$
(3) $b_0 :=$ BLS.Ver(BLS.BKey$((mvk_i)_i, (\sigma_i)_i), eid, \tilde{\sigma}_{eid})$
(4) $b_1 :=$ BLS.Ver(BLS.AKey$((mvk_i)_i), eid||m, \tilde{\sigma}_m)$
(5) return $(b_0 \wedge b_1)$

Figure 8: The stake-based voting scheme Π_{vote}.

message provided. This pair of signatures constitutes a vote which parties immediately share in the network.

Parties collect these votes with the intention of producing a certificate if enough votes (k_v) on the same message are gathered. Each vote received is verified through the VerifyVote procedure, which checks both the eligibility proof and the validity of the signatures. Certificates are produced by running the GenCert procedure, which produces two aggregate signatures: one on the election id and one on the EB header.

Critically, the way the two aggregate signatures are produced is different. Namely, for the aggregate signature that is tied to the eligibility procedure we have to take extra care to ensure that the individual signatures used to produce it are themselves valid, to avoid fake eligibility attacks. Moreover, for performance reasons, we have to do this without directly validating each one of them. To achieve this property, we adopt the batch verification technique presented in [6] (corresponding to procedures BLS.BKey and BLS.BSig), where each individual signature is first raised to an unpredictable exponent and then multiplied, building on ideas from [4]. On the other hand, for the second aggregate signature, we only need to ensure that the parties mentioned in the key-set did indeed tried to produce a signature on the related message, which can be guaranteed by following the classical multiplicative aggregation approach (procedures BLS.AKey and BLS.ASig) This way, parties only have to verify (i) the validity of the two aggregate signatures and (ii) that each individual signature on the VRF proof corresponds to a small hash, resulting in a rather fast certificate verification procedure VerifyCert.

Following the concrete parameter analysis in [6], we can set k_v in the order of 1000 (in the 33% adversary setting), which leads to certificates of size approximately $1000 \cdot (48B + 8B) \approx 56KB$, assuming we are working in the BLS12-381 elliptic curve and each key is associated with an 8 byte pointer–allowing for 2^{64} registered keys in the system. [20] Given that the average block size for Bitcoin, Ethereum, and Cardano is $1MB$, $1MB$, and $90KB$, respectively, a certificate of our scheme easily fits in a block of any one of those blockchains. Moreover, given that parties verify votes as they receive them, we expect certificate generation to be rather fast, as it only requires the computation of two aggregate signatures.

We stress that we present this simple voting scheme to demonstrate the feasibility of our design. We expect that performance improvements can be made along the lines of the techniques presented in [6, 7, 28], e.g., on how eligibility checks are performed to further lower the certificate size. Next, we turn our attention to analyzing the security provided by our scheme.

D.3 Security analysis

Our voting scheme satisfies two main properties: completeness, in the sense that an honest majority of parties can produce a valid certificate for a block it supports, and unforgeability, i.e, that if no honest party supports a block, a malicious minority cannot create a valid certificate for it. To prove the relevant lemma, we rely on the security of the modified BLS scheme presented in [6] which we outline next.

The BLS scheme was shown in [6] to satisfy both unforgeability in aggregate and the stronger batch verification property. In the first one, the adversary cannot create a valid aggregate signature that mentions an honestly generated key, if no individual signature on the same message has been generated by the honest user. The second (stronger) property dictates that the individual signatures that are related to the aggregate signature are themselves correct.

Security of the scheme is proven in the ROM assuming co-CDH on a pairing friendly elliptic-curve is hard. For completeness, we recite the co-CDH hardness assumption.

[20] Remember that keys are pre-registered in the blockchain.

Definition D.1 (The co-Computational Diffie-Hellman Problem). For two groups $_1 = \langle g_1 \rangle, _2 = \langle g_2 \rangle$ of order q, and an adversary $\mathcal{A}$ we define $Adv^{co-CDH}_{1,2}$ as:

$$\Pr\left[a, b \leftarrow \mathbb{Z}_q^2; h \leftarrow g_1^a; t_1 \leftarrow g_1^b; t_2 \leftarrow g_2^b : g_1^{ab} \leftarrow \mathcal{A}(h, t_1, t_2)\right]$$

Assumption D.2 (co-CDH). *We assume $Adv^{co-CDH}_{1,2}$ is negligible for all PPT $\mathcal{A}$ on $_{1,2}$.*

Further, we recite the relevant security properties we are going to use in our analysis. We start with unforgeability in aggregate.

Definition D.3 ([6]). We say that a signature scheme MSP is *unforgeable in aggregate* if any PPT adversary $\mathcal{A}$ wins the following game with only negligible probability.

- The Challenger runs Param $\leftarrow$ Setup(1^κ).
- The Adversary $\mathcal{A}$ selects a number of honest users n.
- The Challenger provides the verification keys mvk_i, and proofs of possession $\boldsymbol{\kappa}_i$ of the honest users to the adversary.
- The Challenger allows the adversary to issue (individual) signing queries on any message and on the behalf of any user.
- The Adversary outputs a tuple $(m^*, \sigma^*, \boldsymbol{mvk}^*, \boldsymbol{\kappa}^*)$.
- The Adversary wins if and only if:
  (1) The vectors $\boldsymbol{mvk}^*$ and $\boldsymbol{\kappa}^*$ have the same length, l.
  (2) MSP.Check$(mvk_i^*, \boldsymbol{\kappa}_i^*) = 1$ for $i \in [l]$.
  (3) There exists at least one index $j \in [l]$ such that mkv_j corresponds to an honest user, and the message m has not been queried w.r.t. mkv_j in the signing oracle.
  (4) MSP.Ver$(m^*, ivk^*, \sigma^*) = 1$, where $ivk^* \leftarrow$ MSP.AKey$(\mathbf{mvk}^*)$.

Theorem D.4 ([6]). *Given Assumption D.2, the BLS signature scheme in Figure 7 is unforgeable in aggregate in the ROM.*

In addition, the following lemma is proven in the same work regarding batch verification.

Lemma D.5 ([6]). *Let $\boldsymbol{mvk} \in_2^n$, and $\boldsymbol{\sigma} \in_1^n$ be two vectors of verification keys and signatures. Then, for any message msg, the check $\prod_i$ BLS.Ver(msg, mvk_i, σ_i) is equivalent to:*

BLS.Ver$(msg$, BLS.BKey$(\boldsymbol{mvk}, \boldsymbol{e_\sigma})$, BLS.BSig$(\boldsymbol{\sigma}))$, for $e_{\boldsymbol{\sigma}} \leftarrow H_p(\boldsymbol{\sigma})$,

except with negligible probability, taken over the outputs of the random oracle H_κ.

Finally, the voting scheme described in [6] shares the same eligibility mechanism as our scheme. Thus, we make use of the eligibility analysis from the same paper to argue about the probability that (i) an honest majority of the stake can create a certificate, and (ii) a dishonest minority cannot create a certificate on its own. To do this, we set the eligibility threshold $T(s)$ in Figure 8 in such a way that the probability that an eligibility check succeeds for a party controlling stake s, i.e., $H(\sigma, i) \leq T(s)$, is equal to $\phi(s) := 1 - (1 - f)^s$, for parameter f; $\phi(s)$ is defined in such a way that breaking the stake into smaller pieces does not help increase the probability of success following [12]. The following corollary is proven in [6].

Corollary D.6 ([6]). *When $f \leq 1/4, \alpha \leq \sqrt{1-f}$, $m_v := \log^2(\kappa)$, $k_v := m_v \cdot \phi(1/2)$, the following hold with overwhelming probability in κ:*

(1) for any distribution of at most $1/2 - \alpha$ stake, less than k_v corresponding voting eligibility checks[21] succeed;
(2) for any distribution of at least $1/2 + \alpha$ stake, at least k_v corresponding voting eligibility checks succeed.

Based on these results we can now prove our main lemma for this section regarding the security of the voting sub-protocol.

Lemma D.7. *Given Assumption D.2, it holds that with overwhelming probability in κ:*

- *for each honestly generated EB at least k_v valid votes are diffused by the end of the corresponding voting phase;*
- *if no honest party supports an EB, then no valid certificate is generated for it.*

Proof. Given that the IBs referenced by honest EBs are received on time by all honest parties with overwhelming probability, for the honest case we only have to show that honest parties with a sufficient amount of stake are eligible to generate a vote for each EB with overwhelming probability. This easily follows by a union bound from Lemma D.6.

Next, we focus on the malicious case. For the sake of contradiction, assume that there exists an adversary $\mathcal{A}$ that can produce a valid certificate for an EB that no honest party supports with non-negligible probability, while controlling keys that correspond to a minority of the stake distribution. We are going to use $\mathcal{A}$ to construct an adversary $\mathcal{A}'$ that breaks the security of the BLS scheme.

$\mathcal{A}'$ is going to run $\mathcal{A}$ while simulating an execution of the blockchain protocol. Initially, it is going to get the honest parties keys from the challenger in the unforgeability in aggregate experiment and set them as the keys of the honest parties in the blockchain execution. Note, that $\mathcal{A}'$ does not know the corresponding secret keys. Then, it is going to perfectly simulate the rest of the execution, by querying the challenger any time an honest party is required to produce a signature, i.e, when it creates a new vote. Finally, if $\mathcal{A}$ at some point produces a valid certificate c that either contains an honest key in the key list for which the challenger has not been asked to produce the related signature, or that does not contain any honest key, $\mathcal{A}'$ outputs the individual signature σ_i corresponding to the honest key in the first case, or the aggregated signature $\tilde{\sigma_m}$ in the second case.

Next, we analyze the success probability of $\mathcal{A}'$. First, note that we simulate the blockchain execution perfectly in the eyes of $\mathcal{A}$, and thus $\mathcal{A}$ is going to create a valid certificate in the reduction with non-negligible probability.

Since the certificate is valid, it must be the case that both aggregated signatures $\tilde{\sigma}_{eid}$ and $\tilde{\sigma}_m$ verify, and that individual signatures in $\boldsymbol{\sigma}$ pash the eligibility check. We split the analysis into two cases: In the first one, assume that no honest key is contained in the list of keys included in the certificate. By Lemma D.6 it is implied that one of the individual signatures in the certificate is invalid. However, by Lemma D.5 the validity of $\tilde{\sigma}_{eid}$ implies validity of the individual signatures presented in the certificate, which leads to a contradiction.

In the other case, at least one honest key is included in the key list of the certificate for which the challenger has not been

[21] As an eligibility check here we count a single hash-then-compare operation.

queried on the certified message. Thus, $\tilde{\sigma}_m$ can be used to win the unforgeability in aggregate experiment.

By a simple averaging argument it follows that $\mathcal{A}'$ breaks either the unforgeability in aggregate property of BLS or the batch verification property outlined in Lemma D.5 with non-negligible probability. The lemma follows. □

E GENERALIZATIONS

E.1 Multi-epoch protocol

To lift the extension protocol to span multiple epochs (reflecting underlying stake change over the course of time), the respective mechanisms of the base protocol can be used (as for instance explicitly given in [12]). Once the underlying base protocol outputs the new stake distribution, that one is simply used for the IB/EB lottery and voting.

E.2 Different consensus resources

Our protocol extension is applicable to protocols based on other resources than stake, e.g., proof of work (PoW). Note, however, that the block/vote lotteries are tightly coupled to the pipeline stages—enabled by the fact that the VRF lotteries are tied to a particular time slot. To match these guarantees in a PoW protocol, this would make it necessary for the PoW to additionally prove that it was not computed ahead of time. Such proofs can be given, e.g., by referencing a stable block in the base protocol (a mechanism used in [33])—but still giving the adversary a high degree of freedom of what "time stamps" to effectively choose (e.g., by referencing a newer, unstable block). To compensate this effect, the pipeline length L would have to grow to levels that would push liveness into intolerable regions.

This problem can be avoided by applying a mechanism in spirit of [19]. For each epoch or pipeline instance a committee is selected by a (fair) PoW lottery based on input endorsing—and assigning a weight to each committee member proportional to its amount of input blocks contributed during the committee selection process. The committee members can then be assigned timed block production and voting rights for the pipelines by VRF lotteries in function of their weights—in the same way as described in this paper.

Note that, as a consequence, the security of the underlying PoW protocol is somewhat degraded as public-key cryptography needs to be introduced, opening it up for new attack vectors (long-range attacks or the disability to recover from adversarial spikes).

F DEFERRED PROOFS

F.1 Lemma 3.3

Lemma 3.3. *Let ε such that $0 < \varepsilon < 2\alpha_{\mathrm{H}} - 1$. A slice is ε-successful except with probability $\varepsilon_{\mathsf{suc}}(L) := O\left(e^{-\varepsilon^2\Omega(f_{\mathrm{I}}L)}\right) + O\left(e^{-\varepsilon^2\Omega(f_{\mathrm{E}}L)}\right)$.*

Proof. The upper bound follows from a union bound over the three properties of a successful slice and the Chernoff-Hoeffding bound as given in Appendix C.

(1) Let $X_{ij} \in \{0, 1\}$ be the indicator random variable that honest party j gets the right to produce an IB in slot i (aka slot leadership), let there be n_H honest parties, where the honest party with index $j \in [1, n_H]$ holds relative stake α_j, and let

$$X \;=\; \sum_{i=1}^{L}\sum_{j=1}^{n_H} X_{ij}$$

be the random variable representing the number of honest slot leaderships during the given slice. Since $\Pr[X_{ij}] = \alpha_j f_{\mathrm{I}}$,

$$E[X] = \alpha_{\mathrm{H}} f_{\mathrm{I}} L\,,$$

and, by the Chernoff-Hoeffding bound from Appendix C, the probability that Property (1) is violated is thus

$$P_1 \;\leq\; \exp\left(-\frac{\varepsilon^2\alpha_{\mathrm{H}} f_{\mathrm{I}} L}{2}\right).$$

(2) In expectation, $f_{\mathrm{I}}L$ IBs are generated during the slice. By the Chernoff-Hoeffding bound, along the lines of the above, the probability that Property (2) is violated is thus

$$P_2 \;\leq\; \exp\left(-\frac{\varepsilon^2 f_{\mathrm{I}} L}{3}\right).$$

(3) In expectation, a slice produces $\alpha_{\mathrm{H}} f_{\mathrm{E}} L$ honest endorser blocks, and at most $(1-\alpha_{\mathrm{H}}) f_{\mathrm{E}} L$ adversarial ones. The probability that there are at least as many adversarial as honest endorser blocks can thus be estimated by a union bound over the probabilities that the honest production undercuts its expectation by a factor of $(1-\varepsilon)$ and the adversarial production exceeds its expectancy by a factor of $(1+\varepsilon)$, as long as $(1-\varepsilon)\alpha_{\mathrm{H}} > (1+\varepsilon)(1-\alpha_{\mathrm{H}})$, which is equivalent to $\varepsilon < 2\alpha_{\mathrm{H}} - 1$. Thus,

$$P_3 \;\leq\; \exp\left(-\frac{\varepsilon^2\alpha_{\mathrm{H}} f_{\mathrm{E}} L}{2}\right) + \exp\left(-\frac{\varepsilon^2(1-\alpha_{\mathrm{H}}) f_{\mathrm{E}} L}{3}\right).$$

□

F.2 Lemma 3.5

Lemma 3.5. *Any IB (indirectly) referenced in a* Vote2*-certified EB from slot s is downloaded by every honest party no later than in slot $s + \nu L$ for*

$$\nu \;:=\; \frac{(\lambda-\rho)(\lambda+\mu+2)+(\lambda+1)\rho}{\rho+1}\,.$$

Proof. Along the lines of the argumentation in the proof of Lemma 3.4, we demand that input block data available at the onset of the Vote1 to any honest party will be downloaded by every honest party within the next ν slices.

In particular, we demand that

$$\tau\left((1+\varepsilon) f_{\mathrm{I}}(2+\lambda+\mu+\nu)L + d\right) \overset{!}{\leq} \nu L\,, \tag{1}$$

This is satisfied if

$$f_{\mathrm{I}} \;\leq\; \frac{\nu-\rho}{(1+\varepsilon)(\lambda+\mu+\nu+2)} \cdot \frac{1}{\tau}\,.$$

Given the choice of f_{I}, this is satisfied if ν is chosen such that

$$\frac{\lambda-\rho}{\lambda+1} \;\leq\; \frac{\nu-\rho}{\lambda+\mu+\nu+2}\,,$$

which requires

$$\nu \;\geq\; \frac{(\lambda-\rho)(\lambda+\mu+2)+(\lambda+1)\rho}{\rho+1}\,,$$

which corresponds to the choice of ν.

□

F.3 Lemma and theorems of Section 4

Lemma 4.1. *Given a sequence of adjacent pipelines, the expected fraction of unsuccessful pipelines is at most* $\varphi_{\text{suc}} = O(\lambda^2)\varepsilon_{\text{suc}}(L)$.

Proof. Slice failures are independent and occur with probability $\varepsilon_{\text{suc}}(L)$. A slice failure can only affect $O(\lambda^2)$ pipelines, thus the lemma follows by a simple union bound. □

Lemma 4.2. *Half of all successful pipelines are covered from the base protocol.*

Proof. First, note that re-inclusion of Vote2-certified EB from other pipelines is more relaxed (what has been seen by the end of the pipeline EB belongs to) than what is required by the base-protocol EB-inclusion rule (what has been seen Δ_{hdr} before the said pipeline). Thus, for each successful pipeline, there is an honest Vote2-certified EB that is eligible for inclusion in the base protocol.

Now, consider any successful pipeline instance i ending at time slot t (and producing an honest Vote2-certified EB_0) and consider any honest payload entering the ledger at time $\tau \in [t+2L, t+2L+\eta]$ (which is guaranteed by ledger quality).

Given the initial observation, and the fact that recursion covers time $\eta + 3L$, said payload either (directly or indirectly) includes EB_0 or another EB from that pipeline— covering at least the pipelines covered by EB_0—or directly includes a Vote2-certified EB from the pipeline instance ending at time slot $t + L$. Only in the latter case, the input blocks of pipeline instance i is lost, and thus the lemma follows. □

Theorem 4.3. *Let* ε *be such that* $0 < \varepsilon < 2\alpha_{\text{H}} - 1$. *Then, Full Leios achieves persistence with parameter* $w' = w$, *and, on average (over all transactions), the protocol achieves liveness with parameters* $r' = u'$, *and* $u' = u + \frac{L}{\varepsilon_{\text{suc}}(L)} + O(\lambda^3)L$.

Proof. Persistence (as a property of the base protocol) remains intact: $w' = w$. Average liveness follows from the fact that the expected wait time for inclusion in a successful pipeline is $\frac{L}{\varepsilon_{\text{suc}}(L)} + L$, the maximal time delay for the pipeline delivery of all IBs is $O(\lambda^3)L$, and that the base protocol satisfies liveness with parameter u'. □

Theorem 4.4. *On average, Full Leios achieves a throughput of*

$$\frac{1}{1+\varepsilon} \cdot \frac{\lambda - \rho}{\lambda + 1} \cdot (1 - 2\varphi_{\text{suc}}) \cdot \alpha_{\text{H}}$$

of honest input-block data relative to C.

Proof. The proof follows from the fact that half of all successful pipelines are covered in the mainchain, and from the rate φ_{suc} of unsuccessful pipelines. □

F.4 Analysis of anti-equivocation measures

Lemma F.1. *For each block opportunity, each honest party downloads at most two block headers and one block body.*

Proof. The lemma follows trivially from the protocol description in Section 5. □

Lemma F.2. *If* EB *is voted by an honest party, then all* IB*s referenced in* EB *are downloaded by all honest parties.*

Proof. See Fig. 4. An honest party only votes for EB if the header of each referenced IB was received by time $s + 2\Delta_{\text{hdr}}$ where s is the slot of IB, and no equivocating header for any such IBs was received by time $s + 4\Delta_{\text{hdr}}$ (middle row). This implies that, for each referenced IB, all honest parties received the header by time $s+3\Delta_{\text{hdr}}$ as the first header seen for the respective block opportunity (bottom row). All honest parties thus 'prefer' and download all IBs referenced in EB. □

Lemma F.3. *If* EB *is voted by an honest party, then* EB *gets downloaded by all honest parties.*

Proof. An honest party only votes for EB if the header of EB was received by time $s + \Delta_{\text{hdr}}$ where s is the slot of EB, and no equivocating header for EB was received by time $s + 3\Delta_{\text{hdr}}$. This implies that all honest parties received the header of EB by time $s + 2\Delta_{\text{hdr}}$ as the first header for the given block opportunity, and thus download EB. □

Lemma F.4. *Given that the respective pipeline is successful, every* EB *produced by an honest party gets voted for by each honest party.*

Proof. The IB timing rule for IB inclusion maintains enough slack to imply the necessary timing (see Fig. 4) for respective voting for every honest party. Similarly, the EB timing rule for voting is implied by the delivery guarantees for EB headers. □