

# Fully Decentralized Collection of Attestations for Single-Slot Finality in Ethereum

János Tapolcai, Bence Ladóczki

Dept. of Telecommunications and Artificial Intelligence, Faculty of Electrical Engineering and Informatics, Budapest University of Technology and Economics, and with the HUNREN-BME Information Systems Research Group.

**Abstract**—After successfully transitioning from proof-of-work to proof-of-stake, the Ethereum blockchain’s developer community has set an ambitious goal of achieving rapid block finalization, ideally completing it before the next block proposal, a concept known as *single slot finality*. Currently, block finalization on the ETH beacon chain takes  $\sim 15$  minutes to collect the attestation from the majority of the validators. The current protocol has several drawbacks, including slow finalization, high bandwidth usage and a rigid aggregation structure that is prone to failures. The challenge is collecting cryptographic signatures from close to a million validators distributed worldwide, connected to the network in 12 seconds without requiring high bandwidth internet connections from the peers in the network. This study presents an alternative scheme that has the potential to realize single-slot finality. Ours is a fully decentralized approach in which no node has a specific role, rendering it more robust than the current one. We simulate our heuristics on the Ethereum network topology and demonstrate that it can efficiently collect a million attestations from almost ten thousand physical nodes.

## I. INTRODUCTION

The success of public blockchains lies in the fact that anyone can join the network. Untrusted peers allocate scarce resources (money, computational power, etc.) to validate new blocks, verify new transactions and monitor the health of the system. Such a mechanism is essential for building trust in cryptocurrencies. When the Ethereum network utilized proof-of-work (PoW) as consensus mechanism, as many as one million people were mining Ether [1]. In September 2022, Ethereum transitioned to a proof-of-stake (PoS) consensus mechanism and this event is known as *The Merge*. In PoS, the goal was to retain this large community of validators, even if it meant slower finalization of blocks. As of December 2024, the number of validators stands at 1 064 863.

In PoW blockchains, it was standard practice to wait for the validation of six blocks before considering a transaction completed. In PoS Ethereum, the block time is referred to as a *slot*, with each slot lasting approximately 12 seconds. Following the transition to a PoS system after the merge, the recommended wait time has increased to a range of 64 to 95 slots (2 to 3 epochs), which translates to roughly 15 minutes. While the 6-slot waiting period in PoW served as a practical guideline, the 64 to 95 slots in PoS provide a theoretical guarantee for transaction finality, as outlined in [2], the difference is significant.

The project was partly supported by Ethereum Academic Research Grant and by Project no. K23 146347 of National Research, Development and Innovation Fund of Hungary.

In a PoS blockchain, a block is deemed finalized when a supermajority, specifically 67% of the validators, has verified it, in a process referred to as consensus. Once a validator verifies a block, it generates and broadcasts a digital signature (also called as *attestation*). As to the current consensus mechanism of the Ethereum network, the Casper protocol [3] orchestrates the collection of attestations. At present, Casper is parameterized such that it does not have to collect supermajority votes in a single slot. Instead, it divides time into *epochs* (32 consecutive slots). Validators are (pseudo-randomly) assigned to particular slots in an epoch, resulting in only a subset of validators – currently 30 586 – participating in the attestation process for each slot. This setup reduces bandwidth usage to about 1/32 of the bandwidth that would be required if all validators were to attest in every slot. However, this efficiency comes at the cost of much longer delays in block finalization.

Rapid block finalization is crucial for ensuring the security and the efficiency of a blockchain. When a block is finalized promptly, it becomes immutable, greatly reducing the risks of chain reorganizations and double-spend attacks. Accelerated finality simplifies operation, and it can lead to a more predictable and reliable system for both users and developers. In PoS blockchains, the time delay from when a block is proposed until it is finalized opens up attack surfaces that could potentially give rise to the coexistence of parallel chains [4], [5]. While such unfortunate situations have not yet been observed – probably due to the liveness of the network maintained by over 95% of honest and active validators – a significant drop in network liveness could lead to serious complications. Concerns associated with long finality periods also include vulnerabilities to maximum extractable value (MEV) exploitation and chain reorganizations, which could undermine the network security and its fairness [6], [7], [8], [9], [10] guarantees. Consequently, the pursuit of achieving single-slot finality is among the most critical technical tasks, attracting considerable attention and effort in the blockchain community [11], [12].

According to our measurements, the daily bandwidth utilization for broadcasting attestation data in the Ethereum network is approximately 9 gigabytes. This level of data consumption is under tolerance levels for many users, especially compared to the bandwidth requirements of operating a full Ethereum node, which requires roughly 20 gigabytes for downloads and 17 gigabytes for uploads every day. In addition to attestation data, the additional bandwidth is required to download new

blocks, to process incoming transactions and to maintain the robustness of the peer-to-peer (P2P) network. Given the substantial commitment required to become an Ethereum PoS validator, marked by a stake of 32 Ether, it is reasonable to expect that such participants have Internet connections capable of supporting these relatively high bandwidth needs. Nevertheless, we are committed to achieving the advantages of single-slot finality without escalating the bandwidth requirements. This goal is vital to ensure broad participation across the network, catering to a diverse community of users, from those relying on mobile connection with its inherent limitations to those in areas with less developed network infrastructure or facing data usage constraints.

The approach implemented in PoS Ethereum involves constructing a randomized aggregation tree. The aggregation structure includes some redundancies, making it more akin to a Directed Acyclic Graph (DAG) than a conventional tree. It has become the de facto method for synchronizing large distributed systems, leveraging state machine replication [13] to overcome the challenges posed by the Byzantine Generals Problem [14]. Similar methods are employed not only in blockchain systems but also in distributed databases (e.g., Google Spanner, CockroachDB) and IoT networks.

The aggregation tree must be randomized to preserve validator anonymity, ensuring that their IP addresses remain hidden. However, a recent study, [15], revealed that even with randomized trees, one can de-anonymize validators with continuous monitoring. Specifically, running only four nodes for three days is sufficient to identify more than 15% of the validators in the P2P network of Ethereum. What further exacerbates the situation is that the study highlighted a significant geographic bias in validator distribution and the frequent use of single machines to host multiple validators. This is partly due to Ethereum’s staking requirement, whereby users can only stake 32 ETH quantities. Stakeholders wishing to stake more have to run multiple validators. A convenient way is to install them on the same machine. The previous study found that only 27% of the validators operated on dedicated machines, while the rest were co-located with other validators. Geographically, hotspots like California showed a high density of validators implying a small physical distance between these nodes.

These findings shed light on the fact that a randomized aggregation tree is inefficient for this use case. Instead of leveraging validators in close physical proximity to aggregate votes, the protocol requires waiting for validators distributed across distant parts of the world at every level of the aggregation process. In such an environment, a fully distributed information collection method, where nodes share their observed data promptly and directly with nearby validators, could be more appropriate for the blockchain scenario. These two approaches – *randomized aggregation trees* and *fully distributed information sharing* – differ fundamentally, and the engineers’ choice between them depends on their suitability for the specific application. This work carries out the necessary steps to properly perform such a decision.

One of the primary challenges is the lack of a direct com-

parison between the current attestation collection mechanism in Ethereum and a hypothetical alternative with millions of validators worldwide. Consequently, this study uses empirical data from the Ethereum network to develop a simulator that models the advantages of an approach that enables the collection of information in a fully distributed way. Simulating a million validators itself poses significant computational challenges, particularly due to the memory constraints of our computing infrastructure. However, simulating at least a million validators (unfortunately there is much more in practice) is essential to draw clear conclusions about our main question: Is a fully distributed approach based on prompt data-sharing techniques more efficient than the current randomized aggregation trees?

We demonstrate that with fully distributed information sharing, the long-awaited single-slot finality can be achieved while maintaining the current operational conditions, such as:

- Preserving the anonymity of validator IPs.
- Limiting bandwidth usage to at most 10-20 gigabytes per day.
- Ensuring resilience against malicious nodes.

This fully distributed approach enables the collection of **32 times more attestations per slot** compared to Ethereum’s randomized aggregation tree implementation.

The main contributions of the paper are as follows:

- to our knowledge, this is the first study on the challenges of a fully decentralized attestation collection in a large PoS blockchain network where no node has distinguished roles,
- we provide a novel data structure for messages with aggregated attestations that can be further aggregated,
- we collected several heuristic approaches developed for ad-hoc and delay-tolerant networks that can be utilized for the attestation collection process,
- we present a simulator environment for evaluation and show that the above goals could be achieved in the real Ethereum network.

The paper is organized as follows. In Sec. II, we overview how attestation collection is done in the current system. In Sec. III, we propose a new alternative approach that can handle a million validators. In Sec. IV, we evaluate the new approach.

## II. THE CURRENT POS PROTOCOL

This section explains how the process of attestation collection is currently implemented in the Ethereum network.

### A. The Peer-to-Peer (P2P) Network

Formally, an Ethereum node comprises two components, an Eth1 client and an Eth2 client. The *Eth1 node* is the original PoW client, sometimes called as the execution client or the execution engine. The client software continues to exist and operate even after the merge. It is mainly responsible for maintaining the original chain and managing the collection and distribution of new transactions via the mempool (a pool of pending transactions). The *Eth2 node* is the PoS client,

sometimes called as the beacon node or the consensus client<sup>1</sup>. This client embodies all the newly introduced functionalities associated with PoS, operating on a distinct chain known as the Beacon Chain. Each validator is connected to an Eth2 node. The Eth1 and the Eth2 clients are connected to two independent peer-to-peer networks, the Eth2 node is to `libp2p` and the Eth1 node to `devp2p`.

The functionalities related to attestations are implemented in `libp2p` leveraging a publish-subscribe (pub-sub) architecture [16]. As to the pub-sub concept, message senders (publishers) do not send messages directly to individual receivers (subscribers). Instead, messages are categorized into channels or topics, and subscribers express interest in receiving messages from specific channels or topics.

It is important to note that Internet routing was originally designed for unicast connections, where the sender and receiver are explicitly defined in the packet header. As such, the Internet does not natively support this kind of pub-sub functionality [17], which implies that implementing pub-sub services requires an overlay network. Overlay networks – by their nature – usually lead to inefficiencies. When performing measurements, we observed a similar situation in the Ethereum network, where broadcasting on a channel results in most Eth2 nodes receiving these messages, regardless of whether they are subscribed to the channel or not.

To preserve the privacy of validators, their IP addresses are concealed using a distributed hash table (DHT) routing protocol in the P2P network. In gossip protocols, participating nodes forward messages without knowing the initiator of the message. And also, this protocol utilizes message relays to enhance privacy and network efficiency.

Overall, hiding the IP addresses is not the primary goal of any DHT routing protocol. They are designed for peer-to-peer networks, eliminating the need for a centralized server. It scales well as the network grows, making it suitable for large distributed systems. Furthermore, it is considered fault tolerant, as DHT is designed to handle nodes frequently joining and leaving the network.

## B. Signature Aggregation in Casper

The protocol designed for collecting attestations in PoS Ethereum is called Gasper. It implements voting by Casper-FFG [3], where validators generate a Boneh-Lynn-Shacham (BLS) [18] signature as a cryptographic proof of attestation. BLS signatures are simply points on an elliptic curve (EC) and two points can always be combined to yield a new point. As such, validators can aggregate attestations. This process involves combining multiple individual signatures into a single, aggregated signature, while maintaining the same level of security. Thus, instead of storing and processing each individual signature, it is sufficient to maintain a single aggregated signature. This approach significantly saves storage space and reduces the time needed for signature verification.

<sup>1</sup>Several implementations exist, such as `Lighthouse`, `Prysm`, `Nimbus` and `Teku`.

For attestation dissemination, 64 pub-sub channels are initialized and each of this corresponds to a so-called *committee*. The validators of each slot are assigned to a committee, and all communication is performed by broadcasting messages in the corresponding pub-sub channel. This assignment is pseudo-random and uses the random number decentralized autonomous organization (RANDAO) mechanism of Ethereum. Every Eth2 node with a validator (or more) subscribes to the corresponding pub-sub channel (or more) and some global pub-sub channels. If it has to participate in the given slot, a validator broadcasts signatures in the pub-sub channel dictated by its pseudo-randomly assigned committee. In each slot and each channel, a total of  $\frac{\mathcal{V}}{64 \cdot 32}$  broadcast events occur, where  $\mathcal{V}$  is the number of validators. For example, currently, for  $\mathcal{V} = 978\,782$  validators, every committee has 478 validators assigned.

In each slot, the validators assigned to that slot generate and broadcast their BLS signature (specific to the block) to the corresponding committee members. In each committee, 16 nodes are pseudo-randomly selected as aggregators. They listen to the corresponding channels and broadcast a signature aggregation message on the global channels. If an attacker were to suppress every aggregator node in a given committee successfully, it would result in the loss of all signatures collected by that committee. Given that there are 16 (redundant) aggregators in each committee, the success probability of such an attack is negligible. The signatures have to be verified prior to aggregation to fend off attacks by malicious nodes.

The *signature aggregation message* contains a bit-vector whose size equals the size of the committee and a single (aggregated) BLS signature. From each committee, the best aggregation message is recorded on the blockchain.

Next, we briefly investigate the performance of the above-explained approach. The first performance indicator we evaluate is the bit size of a signature aggregation message divided by the number of present validators. We call this metric the *average bit size of an ID*, denoted as  $\eta_m$ . A BLS signature is 768 bits in length, and we assume the packet header is 240 bits (with TCP and IP packet headers). We assume that the rest of the fields (slotNumber, committeeIndex, etc.) of a signature aggregation message in Casper take up 256 bits. We note that, this data structure is highly efficient. For example, with  $\mathcal{V} = 978\,782$  validators, the bitfield is only 478 bits long, and when combined with the BLS signature, packet header, and other fields of the message, it results in  $\eta_m = \frac{240+478+768+256}{478} \approx 4$  bits.

The second performance indicator we investigate is the number of bits each node broadcasts, divided by the number of validator signatures known by the node. We call this metric the *communication efficiency* and denote it by  $\eta$ . The daily traffic of 9GB for 30 586 validators results in a communication efficiency of  $\eta = \frac{9 \cdot 2^{33}}{7200 \cdot 30\,586} \approx 400$  bits, where 1GB is  $2^{33}$  bits, and the number of slots per day is  $5 \cdot 60 \cdot 24 = 7200$ .

The difference between  $\eta_m \approx 4$  (ideal) and  $\eta \approx 400$  (actual) is significant and we conjecture that the reason behind it can be traced back to inefficiencies in the signature collection process.

Substantial part of this two-orders of magnitude difference is mainly due to the fact that broadcasting in pub-sub channels – as discussed in [19] – introduces significant bandwidth overhead. Another issue is that consensus clients are engaged in various communication tasks that necessitate the exchange of messages in addition to signature aggregation. These tasks include the dissemination of new beacon chain blocks and managing a local view of the p2p network, which requires activities like topology discovery and incessant HELLO messages. Considering these points, we wish to call attention to significant bandwidth inefficiency in Casper and this issue motivates us to investigate alternative approaches.

### C. Related works

The related theoretical problem [14] was studied in a much more general context. It is referred to as state machine replication [13] in Byzantine fault-tolerant (BFT) consensus protocols. The works provide numerous ideas, but their applicability in a real Ethereum network is questionable because they ignore the fact that the number of network nodes is significantly less than the number of validators. This means that many network nodes have hundreds of validators. If this is not exploited, every protocol becomes very resource-intensive. This can be leveraged by a fully decentralized protocol that does not restrict the communication between validators, thus, it allows for an initial step where validators running on the same computer can exchange information. This step is so efficient that it completely redefines the problem.

Most BFT consensus protocols are limited to a few hundred validators, such as Hot-Stuff [20] and SBFT [21]. Solutions that can handle more nodes tend to be distributed but not fully decentralized, meaning that the nodes have different roles randomly assigned to enhance fault tolerance. These solutions limit the ability of peers in terms of the number and type of connected peers. Typically communication is performed along an aggregation tree, similar to what is implemented in Casper. For instance, in Carnot [22], a binary tree structure composed of committees is adopted. However, none of these solutions are optimized for bandwidth usage, leading us to explore a fundamentally different approach where the protocol dictates the exchange of only a minimal amount of information for peers and their neighbors.

## III. A FULLY DECENTRALIZED APPROACH TO THE COLLECTION OF ATTESTATIONS

We conjecture that the apparent bandwidth inefficiency in Casper is mostly due to the collection of attestations, which happens in a relatively rigid structure, instead of communicating first with those who are easily accessible, such as validators running on the same computer. More specifically, each selected aggregator node has to collect a specific set of signatures at every slot. While this structure enables one to encode the aggregated signatures in an efficient manner the rigidity of this method results in a significant communication overhead. Our high-level idea is that by using a more flexible

data structure one can collect attestations more quickly as it minimizes the need for extensive coordination.

In the current implementation, validators use a data structure comprising bit-vectors for both inter-node communication and inclusion in the blockchain. This approach has its limitations. Aggregated signatures can be further aggregated only if the corresponding bit-vectors are disjoint. If the bit-vectors are not disjoint the aggregated signature contains a signature multiple times, and this cannot be accounted for when using a bit-vector. Casper ensures the issue of disjointness in signature aggregation by allocating each validator to one of the 64 committees. This allocation occurs pseudo-randomly at the start of each epoch, an approach vital for maintaining validator privacy.

When it comes to recording bit-vectors on the blockchain, this approach is less efficient when most bits are set to 1. This happens very frequently with block attestations. In these cases, encoding the positions of the zero bits is a more efficient approach.

We investigate flexible aggregation frameworks wherein a single signature can appear multiple times in an aggregated signature. This feature can be advantageous in scenarios involving multi-level aggregations, where combining two previously aggregated signatures with a common validator is needed. In such cases, it is important to accurately keep track of the multiplicity of each validator’s signature. Below, we refer to the multiplicity as coefficients. These coefficients are ideally 1, in some cases they can be a small integer. In unfortunate cases, the coefficients can be very large without explicit upper limits.

### A. Data Structure for Validator IDs and Coefficients

In the PoS Ethereum network, peers become validators when they lock 32 Ether and this event is recorded on the chain. From this point, the validator is assigned a public key and this key gets recorded in the blockchain. As such the ordered set of validators allows one to identify each of them without relying on their public keys. Formally, the validators are identified by their IDs in the range  $[0, \mathcal{V}]$ , with  $\mathcal{V} \leq 2^{22}$  as per Ethereum 2.0, and it is expected to increase to 1-2 million in the near future. In this section, we present a data structure that can effectively handle validator IDs with coefficients representing validators contributing to an aggregated signature. Recall that the coefficient is usually 1 or a small integer and this is without a reasonable upper limit we can rely on. Our proposed data structure – we introduce below – supports merging operations. The merging operator calculates the union of two sets of IDs and the coefficients that are in the list, corresponding to the aggregated signature.

In a nutshell, the proposed data structure is the following. We maintain a sorted list of IDs, recording the non-negative differences between successive IDs. Zero difference indicates a redundant signature. These differences are stored using prefix codes (variable length codes). We use entropy optimal prefix codes attaining the Shannon compression limit provided that the

frequency distribution of the differences is known a priori. The sorted list allows for efficient merging.

To illustrate the proposed data structure, let us consider the following example. Suppose we have the following set of IDs:

ID:	10	8	2	12	17	18
Coefficient:	1	2	1	1	1	1

First, we sort the IDs in ascending order and replicate each ID according to its coefficient, resulting in:

$$2, 8, 8, 10, 12, 17, 18 .$$

In our data structure, we record the non-negative differences between successive IDs, with the first element being the ID itself. This results in:

$$2, 6, 0, 2, 2, 5, 1 .$$

The list, composed of integers in the range  $[0, 2^{22}]$ , is called the alphabet. Next, we discuss various encoding approaches for the integers in the list. The simplest approach is to use a traditional binary integer encoding on 22 bits. Alternatively, one can use Huffman coding, a variable-length code based on the alphabet frequency. Variable length codes are prefix codes, with shorter codes assigned to more frequent integers and longer codes to less frequent ones. The frequency of the integers (alphabet) can be estimated from historical data. However, Huffman codes require maintaining a lookup table (Huffman tree) that would be extremely large for an alphabet of size  $2^{22}$ . Therefore, we propose the use of Rice-Golomb encoding [23], [24], which was designed for sequences of non-negative integers with geometric distribution.

For Rice-Golomb encoding, the frequency of these integers is estimated using a geometric distribution based on their mean  $\frac{\mathcal{V}}{k}$ , where  $k$  denotes the number of unique IDs. If the actual distribution of the integers follows a geometric distribution, Rice-Golomb coding achieves optimal encoding (with respect to Shannon entropy). In other words, the integers are represented in fewer than 22 bits on average.

The Rice-Golomb coding uses a parameter  $m$ , which in our case can be calculate as:

$$m = 2^{\lceil \log_2(\frac{\mathcal{V}}{k}) \rceil} \quad (1)$$

Here,  $b = \lceil \log_2(\frac{\mathcal{V}}{k}) \rceil$  computes the base-2 logarithm of  $\frac{\mathcal{V}}{k}$  and rounds it up to the nearest integer, ensuring that  $m$  is a power of 2. Such choice of  $m$  attempts to optimize the Rice-Golomb coding for the average value of the integers in the sequence under consideration.

Each integer ID  $i$  is encoded into two parts:

- i) The quotient  $q = \lfloor \frac{i}{m} \rfloor$  is encoded into a unary code, which represents  $q$  by  $q$  1s followed by a zero.
- ii) The remainder  $r = i \bmod m$  is encoded in binary. Since  $m = 2^b$ , the remainder  $r$  can be stored using exactly  $b$  bits.

In the case of  $\mathcal{V} = 20$  and  $k = 5$ , which implies an average integer value of maximum 4, one might use  $b = 2$  in the Rice-Golomb coding, setting  $m = 2^2 = 4$ . Each integer is

then divided by 4, with the quotient encoded in unary and the remainder in binary. The encoding of integers ranging from 0 to 19 with  $m = 4$  is shown in Table I.

For the encoding sequence provided in the original example, each integer would be translated into a Rice-Golomb code with the previously-chosen  $m = 4$ . The encoding begins with the binary representation of  $k$ , the number of IDs, assuming a maximum of  $2^{22}$  IDs, paying attention to IP packet size limits. For example when  $k = 5$ , the 22-bit-long binary representation is 0...0101. In the above example, the encoding sequence would be

$$0 \dots 010 \ 010 \ 1010 \ 000 \ 010 \ 010 \ 1001 \ 001$$

which results in  $22 + 3 + 4 + 3 + 3 + 3 + 4 + 3 = 45$  bits, or on average 6.4 bits per unique ID.

When performing simulations involving a million validator IDs, we observed an average storage requirement of  $\sim 11$  bits per ID, see Sec. IV. When using encoding, we attain approximately 45% space saving, compared to the naive approach of storing validator IDs using a 22-bit binary representation.

### B. The Message Structure for Disseminating Aggregated Signatures

Let us now elucidate the structure of our messages used for broadcasting aggregated signatures. The message is composed of three fields

- 1) the number of IDs in the message encoded in 22 bits,
- 2) the Rice-Golomb encoding of the non-negative differences between successive IDs sorted in ascending order,
- 3) an aggregated BLS signature stored in 768 bits.

*Claim 1:* Two aggregated signature messages can be merged in linear time in proportion to their size.

*Proof:* The union of two ID lists can be calculated in linear time using a merging algorithm [25] operating on IDs sorted in ascending order. The process of aggregating BLS signatures is fast, as it simply involves adding two EC points. ■

Next, we describe a probabilistic model with some assumptions to illustrate the optimality of the proposed message structure that we use here to disseminate aggregated signatures.

Table I: An encoding table for Rice-Golomb codes for  $k = 2$  and  $m = 4$ . The portion before the space in each code (the quotient) is in unary coding (e.g., ‘10’ represents 1). The portion after the space (the remainder) is in binary.

integer	binary code	assumed frequency
0	0 00	0.2
1	0 01	0.16
2	0 10	0.128
3	0 11	0.1024
4	10 00	0.08192
5	10 01	0.065536
6	10 10	0.0524288
⋮	⋮	⋮
19	11110 11	$\approx 0.0000131$

*Claim 2:* The size of the aggregated signature messages is optimal if the following assumptions hold.

- 1) The number of IDs  $k$  in each message can take on any value in the range of  $[0, 2^{22}]$  with uniform distribution.
- 2) The non-negative differences between successive IDs – when sorted in ascending order – follow a geometric distribution with mean  $\frac{\mathcal{V}}{k}$ , where  $\mathcal{V}$  represents the largest ID.
- 3) the BLS signature on curve BLS12-381 provides a proper balance between security and efficiency, thus contributing to the optimality of the signature size [26].

*Proof:* Next, we discuss the optimal encoding of the three components of the aggregated signature message, under certain assumptions. As the optimality of the first and third components directly follows from Assumptions 1 and 3, respectively, we focus on the second field.

For Assumption 2, we assume a geometric distribution for the non-negative differences between successive IDs. The average length of encoding for these differences, using Rice-Golomb coding, is determined by the entropy of the distribution [24].

Formally, given the mean  $\frac{\mathcal{V}}{k}$ , the success probability  $p$  of a geometric distribution is  $\frac{1}{\frac{\mathcal{V}}{k}+1}$ . The encoding length for a difference of  $i$  is then given by

$$\log_2(p(1-p)^i) = \log_2\left(\frac{k \cdot \mathcal{V}^i}{(\mathcal{V} + k)^{i+1}}\right),$$

with the encoding of zero (indicating no difference) being  $\log_2(p)$ . The average encoding length for Rice-Golomb coding is given by :

$$\begin{aligned} \frac{-(1-p)\log_2(1-p) - p\log_2(p)}{p} &= \\ &= \frac{k \log_2\left(\frac{k}{k+\mathcal{V}}\right) - k \log_2\left(\frac{\mathcal{V}}{k+\mathcal{V}}\right)}{k}, \end{aligned} \quad (2)$$

which is according to Shannon entropy, the theoretical lower bound for encoding space under these assumptions. ■

For the previous example, the on-chain storage cost per ID of such a message would be  $\eta_c = \frac{45+768}{6} \simeq 135$  bits. The storage cost per ID decreases if there are more unique IDs included in the message. In unfortunate cases, each duplicated ID requires merely 3 extra bits (i.e., the binary code 000). More unique IDs result in less on-chain storage cost per ID because of two reasons. First, the fixed-size BLS signature is divided by a larger number of IDs, and second, the mean value of the integers stored in the data structure is smaller.

Using the equations in the proof of Claim 2, we can compute the estimated cost per ID ( $\eta_c$  in bits) relative to the number of unique IDs ( $k$ ) in a message used for broadcasting aggregated signatures, see Fig. 1. Messages containing fewer than 20 unique IDs exhibit significant inefficiency. Conversely, the slope of the curve tends to flatten for messages containing more than 200 unique IDs.

Verifying the validity of an aggregated signature message takes a few milliseconds on a commodity computer and involves the following steps:

- 1) Obtain  $k$  as a binary number in 20 bits, and compute  $m$  according to Eq. 1.
- 2) Decode  $k$  integers from the Rice-Golomb encoded sequence, where each code word of an integer consists of a unary-coded quotient followed by a binary-coded remainder modulo  $m$ .
- 3) Convert the decoded differences into a list of IDs by adding them to each other iteratively, starting from an initial value or the smallest ID.
- 4) Retrieve the corresponding BLS public keys for each ID from a lookup table, which is pre-populated with public keys extracted from the blockchain.
- 5) Aggregate the public keys by performing EC point addition on the BLS12-381 curve to obtain the aggregated public key.
- 6) Verify the aggregated BLS signature against the aggregated public key using the appropriate cryptographic verification algorithm.

The last step dominates compute time because BLS signature verification requires the calculation of the Weil pairing (Miller loop). We evaluated a BLS implementation (used by Chia Network [27]) on a commodity laptop (with 2,5 GHz Quad-Core Intel Core i7 CPU) and averaged over 10000 messages and signatures. It took 2.050 ms to verify a BLS signature and 4 ms to perform an EC point addition. It took 265ms to generate a BLS signature.

### C. Conceptual Framework and Underlying Principles

Below, we put forth a fully distributed signature collection approach based on the data structure presented in the previous section. To be resilient against malicious attacks, the collection of the attestation is spread across the network without any selected aggregator nodes. Each node in the network operates independently and can potentially replicate the data and tasks of other nodes, providing redundancy that enhances the system’s resilience to failures and attacks. In this way, there is no specific node that an attacker can target to thwart the collection of attestations. This architecture ensures that if some nodes are dishonest or become unreachable, the overall system can continue to function as usual. Moreover, the distributed approach allows the network to adapt dynamically,

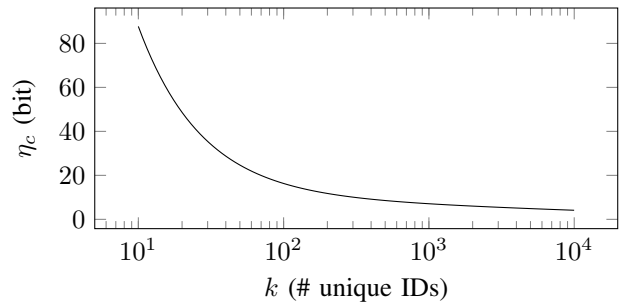


Figure 1: The cost per ID ( $\eta_c$  in bits) with respect to the number of unique IDs ( $k$ ) in the message for disseminating aggregated signatures.

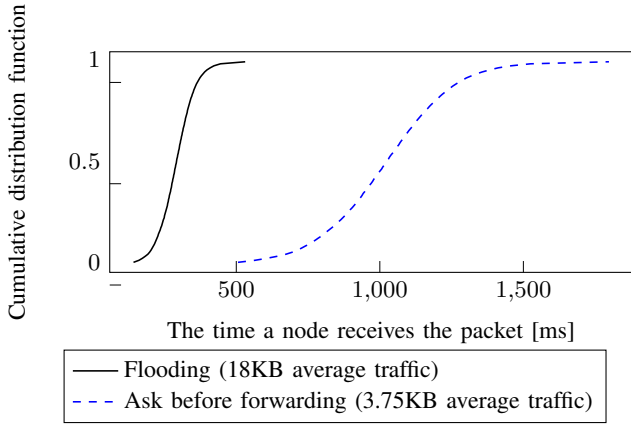


Figure 2: The CDF of the time for each node to receive a 1500byte message to be broadcast in the Ethereum p2p network.

reconfiguring itself in response to dishonest nodes or changes in the network topology. The approach is inspired by the following principles:

Our primary goal is to have *optimal message sizes* with a sufficient number of attestations. Recall that Fig. 1 illustrates the encoding efficiency (cost per ID  $\eta_c$  in bits) relative to the number of attestations (unique IDs  $k$ ) in a message. For example, the encoding is 37 bits per ID if the message contains 10 attestations, while 11.2 bits per ID for 100 attestations, and only 5 bit per ID for 5 000 attestations. The graph representing efficiency flattens after 5 000 IDs. Bear in mind that the maximum size of an IP packet is 65 535 bytes.

No more aggregation is performed once a packet reaches the optimal message size. Instead, large enough messages are broadcast in the network. Broadcast performance is sensitive to certain parameters. For example, the number of neighbors to whom the node forwards the packets. Increasing the number of neighbors leads to quicker dissemination of information but also results in greater bandwidth overhead. We also investigate two versions of this approach. The first is a traditional flooding mechanism whereby the message is dropped if it has been seen before. Otherwise, it is forwarded to all neighbors in the P2P network. The second approach implements a more sophisticated flooding approach, wherein a query message is exchanged before packets are sent to the neighbors. The neighbor node returns whether it has seen the message or not. This can minimize the redundancy of sending duplicate messages to neighbors already informed. On the other hand, it may introduce additional overhead due to the query messages. Fig. 2 shows simulation results when broadcasting a packet of 1 500 bytes in the Ethereum PoS network utilizing both approaches. The first approach generates an average of 18KB of traffic for each node. The second approach can save significant communication data, imposing only an average of 3.75KB of traffic for each node, however it comes with a caveat: broadcast times increase (see the CDF).

Additionally, we implemented several heuristics to *reduce*

*the redundancy of IDs in the messages*. More specifically we approach this issue as follows. Every node in the network tracks the signatures it receives. Recall that these messages include the IDs of the validators along with their corresponding aggregated signatures. Each node maintains a single aggregated signature formed by merging the recently received aggregated signature messages. Time-to-time, each node sends these aggregated signature messages to all of its neighbors, except to those from whom it received the message. A node is considered to have *seen* a signature once it received a valid message containing an aggregated signature with the corresponding ID. This means that the node knows about the signature as part of a combined signature, even if it does not have direct knowledge of the signature itself.

#### D. Attention Dissemination Logic for Nodes

The flow diagram in Fig. 3 explains the flow of processing a message in the proposed approach. When a new message arrives, the node verifies the BLS signature and counts the number of IDs it has not yet seen. Messages containing no new IDs are dropped. Otherwise, the message is stored in a buffer. This buffer stores the incoming messages and there is a timer defined to the buffer to ensure that every message is forwarded after some time has elapsed (`wait_time` parameter). This time is necessary to potentially gather multiple messages in the buffer and this results in a messages with more information (e.g. more IDs). Refer to Fig. 3 for visualization.

If the new message meets certain conditions, the new message is sent out immediately without additional delay. These conditions are when

- (c1) the message arrived has at least `min_sig_num` non-seen IDs, or
- (c2) if the ratio of the non-seen IDs with respect to all IDs in the message is at least `min_sig_perc`, or
- (c3) the number of messages in the buffer is more than `aggr_limit`.
- (c4) the number of non-seen IDs is more than `sig_limit`.

We implemented several heuristics to generate a message from the buffer. In general, we aim to create a message containing the largest number of unseen IDs while minimizing the presence of seen or duplicated IDs. We delve deeper into this topic in the subsequent paragraphs. After the new message is generated, it gets sent out to each neighbor, the message buffer is emptied and the set of seen IDs is updated.

The heuristic keeps track of the IDs for whom the corresponding BLS signature is known. This is achieved by employing standard linear algebra, i.e. Gauss elimination (GE). Ensuring linear independence can potentially save network bandwidth. For instance, consider a scenario where a node initially receives a message containing IDs 1 and 2. If it subsequently receives another message with IDs 1, 2, and 3, the node can apply an EC point subtraction operation to the second message to deduce the individual signature for the validator with ID 3. In this context, the node is said to *know* the signature of ID 3, while IDs 1 and 2 are considered *seen*. In our simulations 0.5% of the BLS signatures were known in

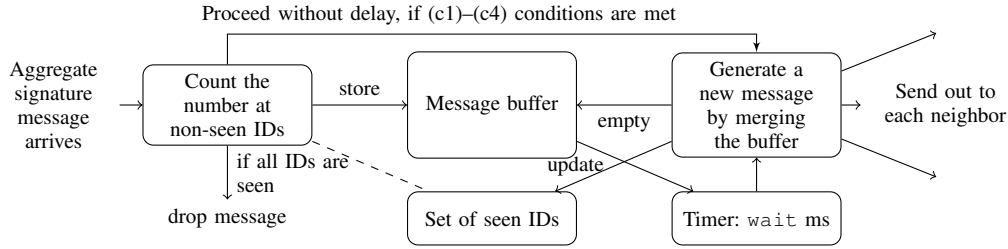


Figure 3: The flow chart of processing an aggregate signature message at each node

average. In the next section, we explain how GE can be used to increase the number of known packets.

### E. GE to increase the number of known packets

Consider a matrix with its rows corresponding to messages and its columns corresponding to validators whose IDs were seen but the respective identity is still unknown. In the first step, we merge identical columns (duplicates are removed), then we also delete duplicate rows. This gives us a matrix with unique rows and columns. We call this the coefficient matrix (CM).

Each row is associated with a message containing a BLS signature. Observe that BLS signatures allow us to perform linear operations just as with coefficients. In other words, if we add two rows together, we can similarly add their corresponding BLS signatures. This process is what we previously referred to as signature aggregation.

Let us briefly explain in detail why we can apply addition and constant multiplication operations on BLS signatures. The BLS signature algorithm relies on two homomorphic one-way functions that map field elements to EC points on two pairing-friendly ECs, forming two distinct cyclic groups. A BLS signature is essentially an EC point on the second curve. An EC point is defined by  $x$  and  $y$  coordinates over a finite field. Furthermore, homomorphic one-way functions [28] imply the existence of an efficient algorithm for addition and constant multiplication operations on EC points. These operations are computationally efficient, requiring only a few operations on the  $x$  and  $y$  coordinates over the finite field, taking mere tens of microseconds in our implementation. Be as it may, we are dealing with a CM with each row corresponding to a BLS signature. In other words, a linear system of equations have to be solved. The right-hand side of the equation system has EC points. Addition and constant multiplication are seamless operations on these objects.

If the CM is invertible, we can compute the value for each column with a coefficient of 1 using GE. During GE, the linear operations are performed directly on the BLS points on the right-hand side. Recall that, during preprocessing, we merged columns that were identical. If a column originally resulted from merging multiple columns, we could compute the aggregated BLS signature corresponding to the validators associated with those columns.

In our application, the method is most efficient when the coefficients transmitted in the packets are as small as possible

– ideally, with coefficients equal to 1. GE helps minimize the coefficients used. However, the method has practical size limitations; inverting very large matrices becomes inefficient. To address this, we limited the inversion process to matrices of size  $100 \times 100$  and smaller.

Let us use the preceding example to extend this strategy to handle sets of IDs. If a new message arrives with IDs 1, 2, 4, and 5, the node can subtract the aggregated signature of IDs 1 and 2 (which is already known) from the new message. This process isolates the message comprising only IDs 4 and 5.

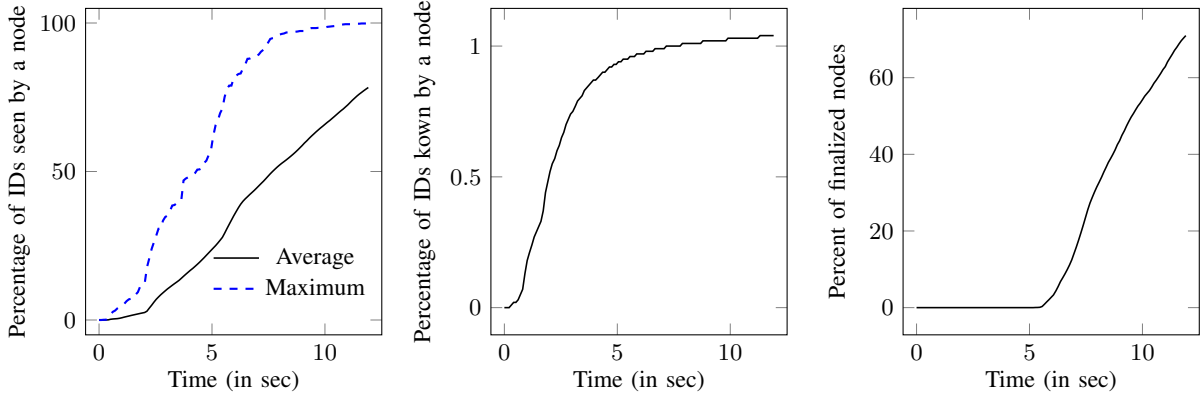
### F. Virtual IDs

The debate to raise the required Ether stake for becoming a validator from 32 Ether to 2048 Ether is ongoing [29]. Furthermore, there is a recent initiative to define clusters of validators for signature aggregation [30]. We call such construction a *virtual ID*. A virtual ID represents a group of validator IDs, and can be registered on-chain as in [30]. The IDs can also be registered in the P2P network. These IDs can be encoded using the method we outlined previously. Registering one or more attached validators under a virtual ID allows a node to conserve bandwidth – a gain that extends beyond the node itself to the entire network.

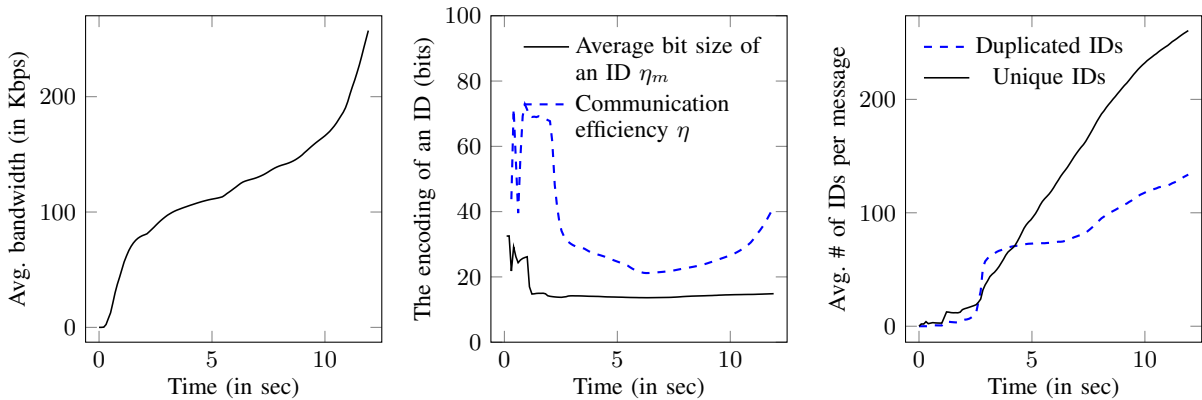
## IV. EVALUATION

Compared to existing solutions this work introduces a fundamentally different approach. The key observation we wish to make here is that certain entities operate multiple validators on the same physical node; and by exploiting this structure, we propose a fully distributed protocol that aims to collect data from the most accessible sources. While fully distributed methods typically offer less scalability compared to structured systems used for signature aggregation—such as those found in Gasper or state machine replication in the BFT literature—they may better reflect the operational dynamics of the Ethereum network. Our central question is whether a fully distributed approach or a rigid but scalable structure proves more effective in this context. To investigate this issue, we developed a discrete event, realistic network simulator using C/C++. The memory management module of the simulator is highly optimized.

We rely on the actual network topology of the Ethereum network to emulate block proposal, signature collection, and aggregation events. Using the Nebula crawler [31], we extracted the `libp2p` topology with 9294 nodes and 934266



(a) The maximum and the average number of signatures seen by the nodes during the slot. (b) The average number of signatures known by the nodes during the slot. (c) The percentage of nodes reaching finality.



(d) The average bandwidth usage of the nodes dedicated to gathering validator attestations during the slot. (e) The bit length of the encoded ID in the messages  $\eta_m$ . The communication efficiency  $\eta$ , which is the total network traffic divided by the average number of IDs seen at a node. (f) The average number of unique IDs in aggregated signature messages, and the number of duplicated IDs in aggregated signature messages.

Figure 4: The performance of the proposed fully distributed signature aggregation scheme in the Ethereum network

links. Since the exact number of validators per consensus client is obscured, we infer it from the count of attestation pub-sub channels a node subscribes to. Nodes with 64 subscriptions are presumed to have a high number of validators, capped at 256. We assigned  $\mathcal{V} = 1\,000\,000$  validators to nodes based on their attestation pub-sub channel subscriptions. Validator distribution across these nodes follows an exponential distribution, as observed in the measurements presented in [15, Fig. 4]. We note that the proposed fully distributed information sharing approach performs better with larger exponents for the exponential distribution. Our estimates are quite conservative and we opted for a smaller decay rate for the exponential distribution used in our simulations.

We estimated the one-way delay for each link using measurements, databases, or by estimating it based on the physical distance of the end nodes. For this, we geo-localized the nodes based on their IP addresses and calculated the geographical distance between them. We aimed to ensure that the one-way delays used in the simulation provide a conservative estimate

of real-world conditions. As a result, the proposed fully distributed information-sharing approach is likely to perform even better in practice than indicated by our results.

At the onset of the simulation, a block proposer chosen randomly and a block is then broadcast in the network. We assume that each node can validate the block in 50ms, produce a valid BLS signature in 2ms, and that peer connections remain stable, with no disconnections occurring during the collection.

Implementing a simulator for the mechanism outlined in Fig. 3 posed significant challenges due to the extensive and necessary state maintenance. Our network comprises 9 294 nodes. Each node manages a substantial number of validators, reaching up to one million. This entails each node maintaining two separate sets: one for seen IDs and another for known IDs, leading to considerable memory demands. At the simulation’s peak, the collective message buffers contained over  $10^8$  distinct messages, further exacerbating memory constraints. The simulation of a mere 12-second window of activity in the crawled Ethereum network required several hours of

wall time on high-performance servers, highlighting the sheer computational intensity of the task.

The crux of the difficulty in the presented work lies in efficiently fitting this vast amount of information into memory. We have fine-tuned our discrete event simulator to address this, focusing on memory optimization. For example, we employed the specialized data structures discussed in Sec. III-B to efficiently store known and seen IDs. Moreover, we utilized a series of straightforward approaches to minimize memory overhead, such as employing smart pointers to avoid unnecessary message duplication and devising a data structure to store only the incremental differences between messages.

We used the following parameters in the simulation; 95% of the nodes use virtual IDs among the ones with at least 2 validators. That is 1905 virtual IDs in total. There are 7833 nodes with validators among 9294 nodes. Aggregating two signatures takes 0.05 milliseconds (ms), signing a message takes 0.5 ms, and verifying a signature takes 2 ms. Once a block is received, it takes 50ms to verify its content in total. If a node has seen 70% of the attestations, it stops forwarding messages. The size of an IP header is 240 bits. Link delays depend on queue lengths.

A node waits (*wait*) at most 700 milliseconds to attain optimal message sizes before forwarding an incoming message with attestations. While waiting, if a new message arrives, the node merges the messages and forwards the combined message at a later time. When either of the following conditions are met, the message is forwarded without delay. This can be `min_sig_num=100` not-seen signatures that have arrived since the last message has been sent out, a new message arrives with `min_sig_per=80%` of seen signatures, or there are already `aggr_limit=8` messages in the buffer, or the total number of signatures in the buffer is more than `sig_limit=5000`.

Fig. 4a shows the maximum and the average number of signatures seen by the nodes in the network. The increase in the average number of seen signatures is linear following the initial first 2 seconds. The maximum number of signatures is also shown in the figure. It reaches 66.7% at 5.2 seconds, which means at least 1 nodes have reached finality.

Fig. 4b shows the average number of signatures exactly known by the nodes in the network. By *exactly known*, we mean that the node possesses the corresponding BLS signature (with coefficient 1). For this to occur, the node must have sufficient information to invert the CM, as detailed in Sec. III-E. In other words, the number of received messages must be at least equal to the number of unknowns, where the unknowns are the sets of seen validators. At the beginning of the slot, the number rises sharply but plateaus relatively quickly as the matrix becomes non-invertible. At the 12th second, it reaches approximately 1%.

Fig. 4c shows the percent of nodes that are aware of finality since the block proposal. At the 12th second, 71% of the nodes have collected 66.7% of the signatures, and thus, they consider the block final. The rest of the nodes are also close to finality as the average percent of seen signatures is 78%

among the network nodes. Keep in mind that the pivotal focus is on confirming that the validator proposing the next block has reached finality.

Fig. 4d shows the average bandwidth usage of a single run of the signature aggregation process. The maximum link bandwidth is set to 6Mbps. The bandwidth usage is very small initially because the messages contain few signatures. Later, we have larger messages, increasing bandwidth consumption. Nodes reach finality and stop participating in the signature collection process, which can be seen after 6 seconds. After 12 seconds, the total bandwidth per node is 3MB, leading to 21GB total traffic per day. Note that the current implementation consumes roughly 9GB of traffic per day, however it only achieves the  $\frac{1}{32}$  of what our method is capable of. The efficiency of this fully decentralized attestation collection approach could potentially be enhanced by allowing the process to terminate once the validator chosen to propose the next block attains finality. We aim to delve deeper into this optimization in our future research endeavors.

Fig. 4e illustrates the average bit size of an encoded ID in the message  $\eta_m$ . Initially, the larger size decreases as the messages contain more IDs. After the first second, the average bit size stabilizes at 14.8. See Fig. 4f for the average number of IDs in the packets. A typical message contains  $\sim 260$  unique IDs, which would be  $\sim 11$  bit per ID in our Shannon entropy-based estimations; see Fig. 1. The relatively small difference underpins the reality of Assumption 2 presented in Claim 2.

We have also quantified the communication efficiency, denoted as  $\eta$ , which represents the total network traffic (only aggregated signature messages) relative to the average number of IDs observed by a node. This efficiency metric is inherently greater than the average bit size of an encoded ID, as the network traffic encompasses not only new IDs but also redundantly transmitted IDs that has already been seen by the node receiving the message. During the slot interval of [2–10] seconds, the discrepancy in  $\eta$  is surprisingly minimal. This indicates that the majority of IDs in the messages are previously unseen, suggesting minimal redundancy. This observation highlights the effectiveness of the method outlined in Fig. 3, demonstrating its strong performance in minimizing unnecessary network traffic.

Fig. 4f shows the average number of unique (the difference is  $> 0$ ) and the number of duplicated (the difference is 0) IDs in the message. While the average number of unique IDs in the message rises throughout the slot, the count of duplicated IDs remains relatively stable.

#### ACKNOWLEDGEMENTS

The authors thank Francesco D’Amato and George Kadianakis for their ongoing contributions through their comments and feedback, which have been invaluable to our work.

#### V. CONCLUSION

In this work we proposed and investigated a fully decentralized approach to collect attestations in PoS blockchains. In our model, every node plays an equivalent role, eliminating

any single point of failure and significantly reducing the vulnerability of the system to targeted attacks. A key element in our approach is an innovative message structure designed for flexible, multi-round attestation aggregation. We show that the employed data structures are time and space optimal considering the specific demands of our protocol. Finally, we conducted a comprehensive simulation on a realistic Ethereum network, comprising 9 294 nodes and 1 000 000 validators, to demonstrate the feasibility of our proposed approach in achieving single-slot finality in 12 seconds. During the simulation, we had to keep track of both the state of each node by maintaining the set of messages they have already received and the set of attestations they have seen. This resulted in  $10^8$  distinct messages in the network at the peak of the simulations. To tackle the sheer amount of messages, we introduced several heuristics and approaches and these methods enabled us to perform comprehensive simulations on realistic network topologies.

Single-slot finality requires an undoubtedly non-trivial change in the protocol, and this cannot be brought forth in a few days; however, we believe that this task can be accomplished step-by-step. In order to underpin this process, we have demonstrated that fully distributed information-sharing approaches are more suitable than the current solution using randomized aggregation trees. We proposed succinct data structures and customized coding algorithms that can significantly enhance the current PoS consensus mechanism in terms of speed and stability. In this work, we illustrated how BLS attestations can be collected in a PoS network using Huffman codes. Furthermore, we backed up our theoretical claims with results from a C++ simulator that models real-world topologies and incorporates actual latency values.

## REFERENCES

- [1] D. Pan, "Ether miners are piling up losses as 'merge' shifts them to altcoins," Bloomberg, September 2022, <https://www.bloomberg.com/news/articles/2022-09-15/ether-miners-are-piling-up-losses-after-shifting-to-altcoins>.
- [2] F. D'Amato and L. Zanolini, "A simple single slot finality protocol for ethereum," Cryptology ePrint Archive, Paper 2023/280, 2023, <https://eprint.iacr.org/2023/280>. [Online]. Available: <https://eprint.iacr.org/2023/280>
- [3] V. Buterin and V. Griffith, "Casper the friendly finality gadget," *CoRR*, 2017. [Online]. Available: <http://arxiv.org/abs/1710.09437>
- [4] J. Neu, E. N. Tas, and D. Tse, "Ebb-and-flow protocols: A resolution of the availability-finality dilemma," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 446–465.
- [5] F. D'Amato, J. Neu, E. N. Tas, and D. Tse, "Goldfish: No more attacks on proof-of-stake ethereum," *Cryptology ePrint Archive*, 2022.
- [6] L. Zhou, K. Qin, A. Cully, B. Livshits, and A. Gervais, "On the just-in-time discovery of profit-generating transactions in defi protocols," 2021.
- [7] B. Weintraub, C. Ferreira Torres, C. Nita-Rotaru, and R. State, "A flash(bot) in the pan: measuring maximal extractable value in private pools," in *IMC '22: Proceedings of the 22nd ACM Internet Measurement Conference*, 2022, pp. 458–471. [Online]. Available: <https://doi.org/10.1145/3517745.3561448>
- [8] A. Obadia, A. Salles, L. Sankar, T. Chitra, V. Chellani, and P. Daian, "Unity is strength: A formalization of cross-domain maximal extractable value," 2021.
- [9] C. F. Torres, R. Camino, and R. State, "Frontrunner jones and the raiders of the dark forest: An empirical study of frontrunning on the ethereum blockchain," in *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021, pp.

- 1343–1359. [Online]. Available: <https://www.usenix.org/conference/usenixsecurity21/presentation/torres>
- [10] J. Piet, J. Fairoze, and N. Weaver, "Extracting godl [sic] from the salt mines: Ethereum miners extracting value," 2022.
- [11] J. Tapolcai, "Flooding protocol for collecting attestations in a single slot," Accessed on September 10, 2025, Nov 2023. [Online]. Available: <https://ethresear.ch/t/flooding-protocol-for-collecting-attestations-in-a-single-slot/17553>
- [12] G. Kadianakis, D. Khovratovich, Z. Zhang, and M. Maller, "Signature merging for large-scale consensus," Accessed on September 10, 2025, Nov 2023. [Online]. Available: <https://ethresear.ch/t/signature-merging-for-large-scale-consensus/17386>
- [13] M. Castro and B. Liskov, "Practical byzantine fault tolerance and proactive recovery," *ACM Trans. Comput. Syst.*, vol. 20, no. 4, p. 398–461, Nov. 2002. [Online]. Available: <https://doi.org/10.1145/571637.571640>
- [14] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM Trans. Program. Lang. Syst.*, vol. 4, no. 3, p. 382–401, Jul. 1982. [Online]. Available: <https://doi.org/10.1145/357172.357176>
- [15] L. Heimbach, Y. Vonlanthen, J. Villacis, L. Kiffer, and R. Wattenhofer, "Deanonymizing ethereum validators: The p2p network has a privacy issue," 2024. [Online]. Available: <https://arxiv.org/abs/2409.04366>
- [16] V. Buterin, D. Hernandez, T. Kampfhefer, K. Pham, Z. Qiao, D. Ryan, J. Sin, Y. Wang, and Y. X. Zhang, "Combining ghost and casper," *arXiv preprint arXiv:2003.03052*, 2020.
- [17] P. Jokela, A. Zahemszky, C. Esteve Rothenberg, S. Arianfar, and P. Nikander, "Lipsin: Line speed publish/subscribe inter-networking," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 4, pp. 195–206, 2009.
- [18] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the weil pairing," in *International conference on the theory and application of cryptography and information security*. Springer, 2001, pp. 514–532.
- [19] D. Vyzovitis, Y. Naporá, D. McCormick, D. Dias, and Y. Psaras, "Gossipsub: Attack-resilient message propagation in the filecoin and eth2.0 networks," *arXiv preprint arXiv:2007.02754*, 2020.
- [20] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, "Hotstuff: Bft consensus with linearity and responsiveness," in *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, 2019, pp. 347–356.
- [21] G. G. Gueta, I. Abraham, S. Grossman, D. Malkhi, B. Pinkas, M. Reiter, D.-A. Seredinschi, O. Tamir, and A. Tomescu, "Sbft: A scalable and decentralized trust infrastructure," in *IEEE/IFIP international conference on dependable systems and networks (DSN)*. IEEE, 2019, pp. 568–580.
- [22] M. M. Jalalzai, A. Mozeika, M. P. Pawlowski, and G. Narayanaswamy, "Carnot: A highly scalable and responsive bft consensus protocol," *arXiv preprint arXiv:2308.16016*, 2023.
- [23] S. Golomb, "Run-length encodings," *IEEE Transactions on Information Theory*, vol. 12, no. 3, pp. 399–401, 1966.
- [24] R. Rice and J. Plaunt, "Adaptive variable-length coding for efficient compression of spacecraft television data," *IEEE Transactions on Communication Technology*, vol. 19, no. 6, pp. 889–897, 1971.
- [25] E. K. Donald *et al.*, "The art of computer programming," *Sorting and searching*, vol. 3, no. 426–458, p. 4, 1999.
- [26] J.-P. Aumasson, D. Kolegov, and E. Stathopoulou, "Security review of ethereum beacon clients," *arXiv preprint arXiv:2109.11677*, 2021.
- [27] "Bls signatures in c++, using the blst library for bls12-381," <https://github.com/Chia-Network/bls-signatures>.
- [28] R. L. Rivest, L. Adleman, M. L. Dertouzos *et al.*, "On data banks and privacy homomorphisms," *Foundations of secure computation*, vol. 4, no. 11, pp. 169–180, 1978.
- [29] M. Neuder, F. D'Amato, A. Asgaonkar, and J. Drake, "Increase the MAX\_EFFECTIVE\_BALANCE – a modest proposal," Civilized discussion furthering Ethereum research, June 2023, <https://ethresear.ch/t/increase-the-max-effective-balance-a-modest-proposal/15801>.
- [30] F. D'Amato and G. Kadianakis, "Netclusters: Speedrunning ssf via networking-layer consolidation," HackMD - Collaborative Markdown Knowledge Base - Ethereum, January 2024, <https://notes.ethereum.org/@fradamt/aggregation-clusters#>.
- [31] D. Trautwein, A. Raman, G. Tyson, I. Castro, W. Scott, M. Schubotz, B. Gipp, and Y. Psaras, "Design and evaluation of ipfs: a storage layer for the decentralized web," in *Proceedings of the ACM SIGCOMM 2022 Conference*, 2022, pp. 739–752.