Esperanza: Efficient and Secure Proof-of-Stake Consensus

April 2019

1 Introduction

A central requirement of cryptocurrencies is that each of the participating distributed nodes should possess and maintain a ledger (chain of transactions) that is consistent with other nodes' ledgers. Cryptocurrencies manage this through an inbuilt *consensus mechanism*, which runs locally on each node. The consensus mechanism takes as input the local view of each node and outputs a blockchain that is (ideally) consistent across all nodes. Consensus in cryptocurrencies is challenging for a number of reasons: (1) networking delays can create different views at different nodes, (2) malicious actors can actively try to thwart consensus, and (3) stringent time or consistency constraints can constrain the classes of viable consensus protocols. Bitcoin's well-known proof-of-work (PoW) consensus mechanism is a notable example that robustly handles malicious actors and network delays; however, this comes at the the expense of latency, throughput, and energy costs. In other works, we showed that if done correctly, PoW can achieve optimal throughput and order-optimal latency. Nonetheless, the energy costs of PoW are difficult to overcome. As such, researchers are actively studying new consensus algorithms for cryptocurrencies.

A leading alternative to PoW is proof-of-stake (PoS), in which a node's influence on the consensus mechanism is proportional to its stake in the system. Like computational work, stake is an asset that is both scarce and easy to verify on the blockchain; this makes it an effective and practical method for gating participation and preventing denial-of-service (DoS) attacks. However, the design of secure PoS schemes with optimal throughput and latency guarantees remains an open question. Key challenges that characterize PoS systems—but not PoW ones—include nothing-at-stake attacks [], long-range attacks [], and the usual questions about how to structure the consensus algorithm (e.g. reliance on voting protocols, finalization, fork-choice rules, etc.). Many papers have studied variants of this problem, with a particular focus on theoretical security guarantees; examples include Ouroboros [**?**, **?**], Thunderella [**?**, **?**], and Algorand [**?**, **?**]. However, the design of a PoS algorithm with simultaneously optimal latency, throughput, and security remains an open question.

We address these properties in a PoS consensus protocol called *Esperanza*. In the design of Esperanza, we have abstracted consensus into three layers: the core consensus layer, the networking layer, and the finalization layer. The *core consensus layer* consists of the algorithms that enable the creation and confirmation of blocks; Nakamoto consensus is a notable example of such an algorithm [**?**]. In Section 3, we present a PoS core consensus algorithm that simultaneously achieves security, throughput, and latency. In many cryptocurrencies, this layer fully defines the consensus protocol. However, in addition to the core consensus layer, we are co-designing a *network layer*; this describes the algorithms by which nodes communicate information to enable consensus. In an efficient system, network performance bottlenecks consensus; it has been recognized by other projects in the space [**?**, **?**]. However, the fundamental limits implicitly assume symmetric algorithms, in which all nodes must receive data before consensus can proceed. This assumption can be broken by co-designing network communication algorithms that account for the structure of the overlying consensus algorithm. We tackle this question in Section 4. The third component of our abstraction is a *finalization layer*. Bitcoin achieves guarantees that are fundamentally probabilistic. While this is adequate for many applications, other regimes may require more definitive guarantees. As such, we discuss in Section 5 a finalization mechanism that provides both Byzantine-fault tolerance and efficiency with respect to the underlying network. Before discussing our algorithmic design at each of these layers, we begin by discussing the requirements of a consensus protocol in Section 2.

2 Design Guidelines

We begin by listing essential requirements for a payment consensus protocol.

1. Permissionless - There should be a low barrier of entry for new nodes to join the consensus process. For example, in Bitcoin, the latest view of the blockchain and a computer are sufficient for nodes to start mining. This property is essential for decentralization.

2. Incentive-compatible - Honest behavior should be a Nash equilibrium (or at least an approximate one), in order to prevent strategic behavior from significantly altering system dynamics. This property is vital for long-term stability against rational actors.

3. Liveness - All honest transactions should eventually enter the main chain of all honest nodes. This property is crucial for protecting against censorship attacks, in which adversarial parties try to prevent other parties from transacting.

4. Consistency - All honest nodes should eventually agree on all transactions.

5. Byzantine fault-tolerance - The system should be robust against a β fraction of Byzantine nodes, or nodes who behave arbitrarily. This is paramount to safeguard the blockchain against attacks from external actors.

Modern vs classic cryptocurrencies: Consensus protocols in classic cryptocurrencies generally consist of straightforward adaptations of ideas from the distributed systems literature. In particular, many of these cryptocurrencies adopts consensus mechanisms that closely resemble either PBFT or an idealized form of leader election. In most of these protocols, communication complexity linearly scales with the number of nodes because they apply one-shot binary consensus for each transaction (event) in isolation. This makes scaling harder as the number of nodes increases in the system. On the other hand, Bitcoin and other modern cryptocurrencies adopt a dynamic perspective of the system and view the transactions as a sequence of events instead of isolated events. They further relax the requirement from binary consensus to *eventual* consensus. These ingenuities reduces the communication complexity by making it independent of the number of nodes and thus avoids the classic cryptocurrencies bottlenecks. In addition, Bitcoin circumvents the 1/3 Byzantine fault-tolerance lower bound and achieves 1/2 Byzantine fault-tolerance.

3 Core Consensus Layer

Notation. We begin by defining some notation. The blocktree is a time-evolving directed tree $G(t) = (V(t), E(t))$. Each vertex $v \in V(t)$ corresponds to a block and $V(t)$ is the set of blocks in the blocktree at time t. Edge $e_{uv} \in E(t)$ if block $v \in V(t)$ is mined over block $u \in V(t)$; u is a parent-block of v (child-block). At time $t = 0$, the blocktree has only a single genesis block i.e, $V(0) = \{b_{\text{genesis}}\}$. The depth of a block b is denoted by $depth(b)$ and is equal to one plus the depth of its parent-block (depth of the genesis block is 0). A block b is defined as

$$b := \{T(b), R(b), pk(b), \text{parent}(b)\},$$

where $T(b)$ is the timestamp of the block, $R(b)$ is the random number of the block, $pk(b)$ is the public key of the miner of the block, and parent(b) is the parent of the block; all these quantities will be defined later. Similar to the bitcoin backbone paper, we ignore the block's payload.

3.1 Protocol

In *Esperanza,* any miner with public key pk and f fraction of coins follows:

Rule 1: For any fixed block b at time $t > T(b)$, if

$$\texttt{Hash}\big(R(b), t, \texttt{pk}\big) < th \times f, \tag{1}$$

the miner is eligible to mine a new block $b_{new} := \big\{t, R(b_{new}), \texttt{pk}, b\big\}$. The resolution of t is in seconds, and th is a fixed threshold which will be adjusted later.

Rule 2: If the miner chooses to mine block b_{new}, it broadcasts b_{new} to the network and stakes P coins. If the block is part of the main chain, the miner gets back its stake and an additional reward of R coins.

Honest strategy: Mine on the longest chain.

We now provide reasoning behind the choice of the above rules.

Source of randomness

In PoW blockchain, a node iterates over different values of *nonces* in search of a valid nonce to mine a new block. The set of valid nonces depends on the transactions in the block. Each block contains a unique set of transactions, and as a result, miners are independently chosen across different blocks. In a nutshell, transactions are the source of randomness in PoS blockchain. This necessitates the nodes to spend computational resources to mine new blocks. As argued in the introduction, PoS blockchains aim at avoiding the use of computational resources to select block miners. Therefore we need a different source of randomness for a PoS blockchain. Having a central beacon for the source of randomness is a plausible solution but it is against the philosophy of blockchain as it would punch a hole in the security; it also has synchronization-related problems which we do not discuss here.

Algorand proposes an elegant solution where each block b is associated with a random number $R(b)$, recursively defined by

$$R(b) := \text{vrf}\big(R(\text{parent}(b)), pk(\text{parent}(b))\big), \tag{2}$$

where vrf stand for Verifiable Random functions []. In this design, the public key of the miners is the source of randomness (instead of transactions). For example, in Rule 1, $R(b_{new}) = \text{vrf}\big(R(b), pk(b)\big)$. In this design, a user's chance of mining blocks is proportional to the number of coins it owns (*asset)* and is independent of its computational power.

Although, this design gives us a decentralized solution of the source for randomness, unlike PoW blockchains, nodes can now mine on multiple chains without splitting their coins (or assets). We address this problem in the next section.

Nothing at stake: The case for a penalty

Strategic nodes maximize the number of blocks mined along the main chain. In Esperanza, a miner can *independently* mine along multiple chains. Specifically, from a miner's point of view, mining on side chains does not decrease its chances of mining on the main chain. Therefore, if not for Rule 2, strategic nodes would mine on all side chains since any of these side chains could potentially become the longest chain in future. This would result in a chaotic behaviour and sabotage the blockchain system. On the other hand, PoW protocols do not face this problem because miners incur *expensive* computation costs (electricity, hardware) which incentivizes them to mine only along the main chain i.e, follow the honest strategy. Rule 2 of Esperanza penalizes miners for mining blocks on side chains and hence incentivizes strategic nodes to follow the honest strategy. The value of the penalty P is designed to mimic the cost incurred by mining on side-chain in PoW blockchain.

Rule 2 solves nothing-at-stake problem for strategic nodes. However, it does not safeguard the system from nothing-at-stake attacks from Byzantine nodes. In the next section, we analyze the strength of Byzantine nodes and slightly modify Rule 1 to obtain 49% Byzantine fault-tolerance.

3.2 Byzantine Adversaries

According to Rule 1 (3.1), for a *fixed* block b, a node evaluates equation (1) at every second to check its mining eligibility. Since each of these evaluations is independent, the time taken for a node to a mine on any fixed block is an exponential random variable with a rate proportional to the fraction of coins it owns. Let the threshold th in equation (1) be adjusted such that the expected rate of growth of a single chain is 1 block per second if all nodes are mining on it via honest strategy. Let the adversary have β fraction of coins.

If the random parameter of the block evolves according to equation (2), the optimal strategy for the adversary is to grow multiple chains by mining on *all* the blocks *all* the time.

Proof. Mining on multiple chains (or blocks) does not effect the chance of mining a new block on any particular chain. On the other hand, since each block has an independent random number, mining on each of them is independent, and therefore, mining on all blocks can only increase the longest chain. □

Trivially, the longest chain of the adversary grows at least at the rate of β. We now show that under the optimal strategy, the longest chain of the adversary grows at a rate at most $e\beta$. We extend the earlier notations 3 to prove our result:

Extended Notations: As before, the blocks are represented by a time-evolving directed tree $G(t) = (V(t), E(t))$. The adversary is growing a private chain over block b starting at time $t = 0$, and without loss of generality let the block b be the genesis block $V(0) = \{b_{\text{genesis}}\}$. At all time t, the adversary follows the optimal strategy and independently mines on *all* the blocks for new child-blocks. From the point of view of a fixed block, its child-blocks are generated as a Poisson point process with rate β i.e the time interval between the child-blocks is an exponential random variable with rate β. Let the depth of the tree G denoted by $D(t)$ and defined as the maximum depth of its blocks. Let the random variable $X_i(t)$ denote the number of blocks at depth i. The rate of growth of the longest chain is denoted by $r_{\text{adv}} := \frac{D(t)}{t}$; trivially $r_{\text{adv}} \geq \beta$.

Under the strategy (3.2), the longest chain of an adversary with β fraction of coins grows at the rate $r_{adv} \leq e\beta + o_t(1)$. Here $o_t(1)$ tends to zero as time t goes to infinity.

Proof. We first derive the expectation of $X_i(t)$. Let $X_i(t, t') = X_i(t) - X_i(t')$. For very small values of δ, we have

$$\begin{aligned}
&\mathbb{P}\big(X_{i+1}(t+\delta,\ t) = 1\big) = \beta\delta X_i(t) - o(\delta)\\
&\mathbb{P}\big(X_{i+1}(t+\delta,\ t) = 0\big) = 1 - \beta\delta X_i(t) + o(\delta)\\
&\mathbb{P}\big(X_{i+1}(t+\delta,\ t) = k\big) = o(\delta^{k-1}) \qquad \forall k \geq 2.
\end{aligned}$$

Therefore, $\mathbb{E}\big[X_{i+1}(t)\big]$ satisfies the following differential equation Check: Interchanging differentiation and integration. May be prove finiteness?

$$\frac{d\mathbb{E}\big[X_{i+1}(t)\big]}{dt} = \beta\mathbb{E}\big[X_i(t)\big] \qquad \forall i \in \mathcal{I} \tag{3}$$

with initial conditions $X_i(0) = 0\ \forall\ i$. The root block satisfies

$$X_1(t) = \mathbb{E}\big[X_1(t)\big] = \begin{cases} 1 & t > 0 \\ 0 & t \leq 0 \end{cases}. \tag{4}$$

By recursively solving equation (3) and (4), we obtain

$$\mathbb{E}\big[X_{i+1}(t)\big] = \frac{(\beta t)^i}{i!}. \tag{5}$$

From the above equation (5) we observe that the expected number of blocks at a fixed depth i, grows with time t. On the other hand, for a fixed time t, the expected number of blocks tend to zero as the depth increases. Both these observations satisfy one's intuitions.

We will use equation (5) to derive the longest chain growth rate. For small $\epsilon > 0$, let

$$i_u := \big(1 + \frac{\epsilon}{\sqrt{i_u}}\big)e\beta t. \tag{6}$$

Since $X_i(t)$ are positive random variable, applying Markov's inequality gives us

$$\begin{aligned}
\mathbb{P}\big(X_{i_u}(t)! = 0\big) &\overset{(a)}{=} \mathbb{P}\big(X_{i_u}(t) \geq 1\big) \leq \mathbb{E}\big[X_{i_u}(t)\big] \\
&\overset{(b)}{=} \frac{(\beta t)^{i_u}}{i_u!} \\
&\overset{(c)}{<} \big(1 + \frac{\epsilon}{\sqrt{i_u}}\big)^{-i_u} \left(\frac{i_u}{e}\right)^{i_u} \frac{1}{i_u!} \\
&\overset{(d)}{<} \big(1 + \frac{\epsilon}{\sqrt{i_u}}\big)^{-i_u} \frac{1}{\sqrt{2\pi i_u}} \\
\mathbb{P}\big(X_{i_u}(t)! = 0\big) &< e^{-\epsilon\sqrt{i_u}}
\end{aligned}$$

Equality (a) follows from the fact that X_{i_u} is an integer. The second equality (b) follows from equation (5). Inequality (c) is obtained by substitution t using equation (6). Inequality (d) uses Stirling's approximation.

Since $X_{i_u} = 0$ with high probability, by definition the tree depth, $D(t) \overset{whp}{<} i_u = \big(1 + \frac{\epsilon}{\sqrt{i_u}}\big)e\beta t$ and this give us $\frac{D(t)}{t} \overset{whp}{\leq} e\beta t$. □

Theorem 3.2 upper bounds the growth rate of adversarial longest chain by $e\beta$. It turns out that this bound is tight for the optimal strategy and this will be proved later.

Modifying the source of randomness

Theorem 3.2 shows that the adversary with β fraction of stake can privately grow a chain at a boosted rate of $e\beta$. On the other hand, the penalty via Rule 2 was specifically designed to over come this behaviour among the strategic nodes (Def. 3.1), and as a consequence they do not receive any such boost. It is to be noted that the adversary does not have to worry about this penalty because it grows the whole tree in private and only reveals the longest chain to public.

The origin of this asymmetry stems from evolution of randomness defined as per equation (2). Since blocks have independent random numbers, the adversary *independently* mines over *all* blocks at *all* times. Mathematically this implies that for the adversarial chain, the rate of growth of number of blocks at depth $i+1$, $\frac{d\mathbb{E}\big[X_{i+1}(t)\big]}{dt}$, is equal to $\beta\mathbb{E}\big[X_i(t)\big]$; whereas the corresponding rate of growth for the public chain is only $\beta \max\big(\mathbb{E}\big[X_i(t)\big] - \mathbb{E}\big[X_{i+1}(t)\big]\big)$. Now that we have identified the origin of the asymmetry, we alleviate it by adding dependency in the evolution of randomness. Let $m \in \mathbb{Z}$ and let k a design parameter.

$$R(b) := \begin{cases} \text{vrf}\big(R(\text{parent}(b)), pk(\text{parent}(b))\big), & \text{if } depth\big(b\big)\%k = 0, \\ R(\text{parent}(b)), & otherwise \end{cases} \tag{7}$$

Block b is a godfather-block if $depth(b)\%k = 0$. From Equation (7) we see that the randomness of a block changes only at *godfather-blocks.* In other words, for $m \in \mathbb{Z}$, blocks along a chain at depths $\big\{mk, mk+1, mk+2, \cdots, mk+k-1\big\}$ share a common random number. Two blocks are siblings-blocks if they have the same parent block.

If the random numbers of the blocks evolve as per equation (7), then the optimal adversarial strategy is to mine multiple siblings godfather-blocks. Mining a sibling to a non-godfather-block does not increase the growth rate of the longest chain.

Proof. Sibling non-godfather blocks share a common random number and thus 'mining events' on these blocks are completely dependent. As a result, the longest chain originating from a particular non-godfather block b is always longer (or same length) than the longest chain originating from its younger sibling (a sibling block mined after block b). Thus mining a sibling to a non-godfather block does not increase the growth rate of the longest chain. However, sibling god-father blocks have independent random numbers and thus mining multiple such block increase the growth rate of the longest chain. □

Under the strategy 3.2, the longest chain of an adversary with β fraction of coins grows at the rate $r_{adv} \leq \big(\frac{ke}{ke+1-e)}\big)\beta + o(1)$.

Proof. Similar to the proof of Theorem 3, we will first derive the expectation of $X_i(t)$. Compared to Strategy 3.2, Strategy 3.2 only mines multiple godfather sibling blocks. Therefore the growth rate of blocks at level $i = mk + l$ (for $m, l \in \mathbb{Z},\ l < k$) depends on $i\%k = l$. In particular we have

$$\frac{d\mathbb{E}\big[X_{mk+l}(t)\big]}{dt} = \begin{cases} \beta\mathbb{E}\big[X_{mk+l-1}(t)\big] & \text{for } l = 0 \\ \beta\mathbb{E}\big[X_{mk+l-1}(t)\big] - \beta\mathbb{E}\big[X_{mk+l}(t)\big] & \text{for } l \in \{1, 2, \cdots k-1\} \end{cases} \tag{8}$$

with initial condition $X_{mk+l}(0) = 0$ and the root block satisfying

$$X_0(t) = \mathbb{E}\big[X_0(t)\big] = \begin{cases} 1 & t \geq 0 \\ 0 & t < 0 \end{cases}.$$

We solve for $\mathbb{E}\big[X_{mk+l}(t)\big]$ via Laplace transform. We first transform the differential equations (8) in Laplace domain, obtain a closed form solution for $\mathbb{E}\big[X_{mk+l}(s)\big]$. After that we transform the equations back to time domain.

First lets apply Laplace transform for the above system of equations (8) (Change notations for laplace equations)

$$s\mathbb{E}\big[X_{mk+l}(s)\big] = \begin{cases} \beta\mathbb{E}\big[X_{mk+l-1}(s)\big] & \text{for } l = 0 \\ \beta\mathbb{E}\big[X_{mk+l-1}(s)\big] - \beta\mathbb{E}\big[X_{mk+l}(s)\big] & \text{for } l \in \{1, 2, \cdots k-1\} \end{cases} \tag{9}$$

Solving the above system of equations gives us (add l term below)

$$\mathbb{E}\big[X_{mk}(s)\big] = \frac{\beta^m}{s^{m+1}(1 + s/\beta)^{m(k-1)}}.$$

The RHS of above expression has a messy denominator term. We use partial fractions to decompose the RHS as a summation of 'simple' fractions terms which can be then transformed to time domain using standard inverse-Laplace transform identities []. Decomposing the RHS of the above equation gives us

$$\mathbb{E}\big[X_{mk}(s)\big] = \sum_{i=0}^{m} (-1)^i \binom{m(k-1)}{i} \frac{\beta^{m-i}}{s^{m+1-i}} + (-1)^m \sum_{j=0}^{mk} \binom{m(k-1)-j}{m-1} \frac{1}{(1+s/\beta)^{m(k-1)-j}}$$

Taking the inverse Laplace transform of each of the individual terms give us

$$\mathbb{E}\big[X_{mk}(t)\big] = \sum_{i=0}^{m} (-1)^i \binom{m(k-1)}{i} \frac{\beta t^{m-i}}{(m-i)!} + (-\beta)^m \sum_{j=0}^{mk} \binom{m(k-1)-j}{m-1} \frac{\beta t^{m(k-1)-j-1}}{(m(k-1)-j-1)!} e^{-(m(k-1)-j-1)\beta t}. \tag{10}$$

We have successfully solved the system of equations (9) to obtain an expression for $\mathbb{E}\big[X_{mk}(t)\big]$. The second summation term of this expression is negligible. We will now obtain a closed form solution for the first term

$$\begin{aligned} \mathbb{E}\big[X_{mk}(t)\big] &= \sum_{i=0}^{m} (-1)^i \binom{m(k-1)}{i} \frac{\beta t^{m-i}}{(m-i)!} + o_t(1) \\ &= \frac{1}{m!} \sum_{i=0}^{m} (-1)^i \binom{m}{i} \beta t^{m-i} (mk-m)(mk-m-1)\cdots(mk-m-i+1) + o_t(1) \\ \text{(Approximation)} &\overset{(a)}{\lessapprox} \frac{1}{m!} \sum_{i=0}^{m} (-1)^i \binom{m}{i} \beta t^{m-i} (m(k-1))^i + o_t(1) \\ &= \frac{(\beta t - m(k-1))^m}{m!} + o_t(1). \end{aligned} \tag{11}$$

Now that we have obtain a closed form solution for $\mathbb{E}\big[X_{mk}(t)\big]$, we apply Markov's inequality for $m_u := \big(1+\frac{\epsilon}{\sqrt{m_u}}\big)\big(\frac{1}{e}+k-1\big)^{-1}\beta t$ to obtain

$$\begin{aligned}
\mathbb{P}\big(X_{m_u k}(t)! = 0\big) = \mathbb{P}\big(X_{m_u k}(t) \geq 1\big) &\leq \mathbb{E}\big[X_{m_u k}(t)\big] \\
&= \frac{(\beta t - m(k-1))^{m_u}}{i_u!} \\
&< \big(1+\frac{\epsilon}{\sqrt{m_u}}\big)\left(\frac{m_u}{e}\right)^{m_u}\frac{1}{m_u!} \\
&< \big(1+\frac{\epsilon}{\sqrt{m_u}}\big)^{-\epsilon m_u}\frac{1}{\sqrt{2\pi m_u}} \\
\implies \mathbb{P}\big(X_{m_u}(t)! = 0\big) &< e^{-\epsilon^2\sqrt{m_u}}.
\end{aligned}$$

The inequality (a) has to be proved. Technically, we need it to be true for $t = m(k-1) + m/e$ Therefore by definition of depth of the tree T, $D(t) \overset{whp}{<} km_u = \big(1+\frac{\epsilon}{\sqrt{m_u}}\big)\big(\frac{1}{e}+k-1\big)^{-1}k\beta t$,and thus we have our required result

$$\begin{aligned}
\frac{D(t)}{t} &\overset{whp}{<} \big(\frac{1}{e}+k-1\big)^{-1}k\beta \\
&\overset{whp}{<} \frac{ek}{ek+1-e}\beta
\end{aligned}$$

□

3.2.1 Security for Nakamoto's Attack

For Bitcoin, Nakamoto proved that an adversary with $\beta < 1/2$ hashing power cannot double spend via growing a longer private chain - Nakamoto attack. In this section, we derive similar results for Esperanza.

In Esperanza blockchain, an adversary with $\beta < \frac{1}{1+e}$ fraction of coins cannot double spend via the Nakomoto attack.

Proof. The theorem in Bitcoin paper [?] relies on the fact that in expectation the public chain grows faster than the adversary's private chain. For $\beta < \frac{1}{1+e}$, we will precisely show that.

For $\beta < \frac{1}{1+e}$, the adversary's longest chain grows at rate less than $\frac{e}{1+e}$. Whereas the public (honest) chain grows at rate $1-\beta$, which is greater than $\frac{e}{1+e}$. Thus, in expectation public chain grows faster than the private chain. □

Note that Lemma 3.2.1 proves the adversary cannot double spend via a Nakamoto attack; however, it does not rule out other possible sophisticated attacks.

4 Network Layer

4.1 Information Load Balancing

We start with the basic setup:

- There are n nodes on a complete graph. Communication on the graph is pairwise and occurs with delay exponentially distributed with mean D. The random delays across each pairwise edge is independent, and also over time (discussed next).
- Discrete arrival process at deterministic intervals: block t arrives at time $t-1$ at a node uniformly randomly chosen. Here t is indexed by positive integers $1, 2, \ldots$. The chosen node is called the *proposer* of block t and the randomness in choosing the proposer is independent across time and other sources of randomness in the model.

- Each node has its own local view of the block tree which is visible to themselves. The root of all the local views is common and called “Genesis".

- Upon arrival of a block numbered t, the proposer attaches block t as a leaf to its local tree. The manner of attachment depends on the fork choice rule. We consider two prominent fork choice rules:

  Nakamoto protocol (Longest chain) The proposer attaches the block t as a leaf to the *longest chain* in the sampled node's *local* block tree.

  GHOST The proposer attaches the block t as a leaf to the *heaviest subtree* in the sampled node's *local* block tree. Here, the heaviest subtree is defined as the subtree with the most nodes.

  An attachment protocol belongs to the family of *local attachment protocols* if the proposer makes the decision on where to attack the block solely based on its *local* tree with no *delay* and *symmetric* with respect to the nodes symmetric with respect to what? this term is overloaded, so it's not clear what is meant. Apparently, both the Nakamoto and the GHOST protocols belong to this general family. (apparently is a weird choice of word)

- Broadcasting rule: The proposer then broadcasts the information (Block t, pointer to Parent block of t) to all the other nodes $i \in [n]$. This communication takes a random amount of time which is exponentially distributed with mean D, for each recipient node independently.

- Updating rule: Upon receiving a message (Block t, pointer to Parent block of t), a node i updates its local view as follows:

  1. The proposer checks if it already has block t's parent block .
  2. If not, it stores block t in a cache of orphan blocks.
  3. If so, it attaches t to its parent. After attaching a block to the local tree, the proposer checks all blocks in the orphan cache to see if any of them can be appended to the local tree. If so, it appends them as well.

For any *local attachment protocol*, one can consider the union of the local trees at each node and define the global tree as follows.

[Global tree] We define the *global tree* at time t of a *local attachment protocol*, denoted as G_t, to be a graph whose edges are the unions of the (Block j, pointer to Parent block of j), $1 \leq j \leq t$ with the vertices being the union of all blocks numbered from 1 to t, together with the genesis block.

Notice that G_t is random and each of the local view at the n nodes is a subset of G_t. Our goal is to understand how G_t evolves. Specifically, we are interested in characterizing how close it is to a chain. The closer G_t is to a chain, the less the “forking" associated with the block chain. Notice that if $D \ll 1$ then G_t is a chain with high probability. On the other hand, if $D \gg 1$ then G_t is a star with high probability. In this section we are interested in understanding how D affects the forking nature of the global tree G_t.

4.1.1 Effect of ℓ-polling

Concretely, the ℓ-polling strategy works as follows: upon arrival of a block t, the proposer of block t selects $\ell - 1$ distinct nodes in the network uniformly at random, and inquires about their local tree. The proposer aggregates the information from the $\ell - 1$ other nodes; here aggregation means the following: if a block's parent is already received, then the block can be appended to the parent. Otherwise, the block is added to the orphan pool until its parent block arrives. The proposer then makes a decision on where to attach block t based on the *local attachment protocol* it follows. One key observation is that there is no conflict between the local trees of each node, so the ℓ-polling strategy simply merges totally ℓ local trees into its union tree. To simplify the analysis, we assume that the polling operation and local tree information feedback from other nodes happen instantaneously. this seems like a really strong assumption! We also assume that each node processes the additional polled information real time with no storage. In other words, the information a node polled at time t is forgotten at time $t' > t$.

The following theorem quantifies the effect of ℓ-polling on the probability of observing a certain tree structure. For any local attachment protocol as defined in the problem setting, denote the proposer for block

j as m_j, and write the event of observing a certain global tree structure with the node sequence $\{m_j\}_{j=1}^t$ as $\cap_{j=1}^t E_{j,m_j}$, where E_{j,m_j} denotes the requirement on node m_j. Then, the probability of observing this tree structure with the vanilla strategy is given by

$$p_1 = \mathbb{E}[\prod_{j=1}^t \mathbb{1}(E_{j,m_j})|\text{all } \{m_j\}_{j=1}^t \text{ are distinct}] + \delta_1 \tag{12}$$

$$= F_t(D) + \delta_1, \tag{13}$$

where $|\delta_1| \leq \frac{t(t-1)}{2n}$. Here $F_t(D) = \mathbb{E}[\prod_{j=1}^t \mathbb{1}(E_{j,m_j})|\text{all } \{m_j\}_{j=1}^t \text{ are distinct}]$ is some function of D. why not state the theorem as an inequality and define δ_1 exactly? It's slightly misleading at first, because it looks like the thm characterizes p_1 exactly

Then, with ℓ-polling, the probability of observing that tree structure can be shown to be

$$p_d = F_t(D/\ell) + \delta_\ell, \tag{14}$$

where $|\delta_\ell| \leq \frac{\ell t(\ell t-1)}{2n}$.

In other words, the effect of ℓ-polling is approximately changing the mean network latency from D to D/ℓ, for n large enough.

Proof. Now we prove Theorem 4.1.1. With the vanilla strategy, we have

$$p_1 = \mathbb{E}[\prod_{j=1}^t \mathbb{1}(E_{j,m_j})] \tag{15}$$

$$= \mathbb{E}[\mathbb{E}[\prod_{j=1}^t \mathbb{1}(E_{j,m_j})|\{m_j\}_{j=1}^t]] \tag{16}$$

$$= \mathbb{P}\left(\text{all } m_j \text{ are distinct}\right) \mathbb{E}[\prod_{j=1}^t \mathbb{1}(E_{j,m_j})|\{m_j\}_{j=1}^t] + \mathbb{P}\left(\text{not all } m_j \text{ are distinct}\right) \mathbb{E}[\prod_{j=1}^t \mathbb{1}(E_{j,m_j})|\{m_j\}_{j=1}^t]. \tag{17}$$

I'm assuming the conditioning in these two expectations is not the same? I.e., the first one is conditioning on sequences of proposers that are all distinct, whereas the second is conditioning on proposers that are not?

We argue that if all the m_j are distinct, then the expression

$$\mathbb{E}[\prod_{j=1}^t \mathbb{1}(E_{j,m_j})|\{m_j\}_{j=1}^t] \tag{18}$$

does not depend on the specific sequence $\{m_j\}_{j=1}^n$. Indeed, due to the symmetry assumption of the protocol, if all the proposers are distinct, whenever one makes the decision about block t, the sole information it is using is the information for the previous $t-1$ blocks, which were sent by its proposers at time $1, 2, \ldots, t-1$, respectively. The delays for these blocks are independent of the identities of the proposers, $\prod_{j=1}^{t-1} \mathbb{1}(E_{j,m_j})$.

It follows from the birthday paradox compution [**?**, Pg. 92] that

$$\mathbb{P}\left(\text{all } m_j \text{ are distinct}\right) \geq 1 - \frac{t(t-1)}{2n}, \tag{19}$$

which implies the bound.

In the ℓ-polling case, since we are conducting sampling without replacement was this stated in the model?, there is higher probability in sampling distinct ℓt nodes compared with the sampling with replacement bound wouldn't the probability be 1?, we use the birthday paradox computation with ℓt persons, and observe that for each block arrival, if for each block $j, 1 \leq j \leq t$, its proposer and the nodes its proposer polled has never proposed or been polled in the past, aggregating the information from ℓ nodes together is equivalent to reducing the network latency mean parameter from D to D/ℓ, since the minimum of ℓ exponential random variables with mean D follows exponential distribution with mean D/ℓ. I don't understand this statement □

4.1.2 Probability of observing the chain with ℓ-pooling

It turns out that when the tree structure is a chain, we have refined results for both the Nakamoto and GHOST protocols.

For both the Nakamoto and GHOST protocols, if the polling parameter ℓ satisfies

$$\ell \geq D\left(\ln t - \ln\ln\frac{1}{\delta}\right), \tag{20}$$

then the probability of the chain Gen $- 1 - 2 - \ldots - t$ happens with probabilty $\delta + o(1)$ as $t \to \infty$ and $n \gg (\ell t)^2$. Here $\delta \in (0,1)$ is the confidence parameter.

Proof. Denote the arrival time of the information (Block j, Point to the Parent of Block j) to node i as $R_{j,i}$. Note that for any i that is not the proposer of block j, $R_{j,i} - (j-1)$ follows exponential distribution with mean D. For any i that is the proposer of block j, $R_{j,i} = j-1$. Denote the proposer for block j as $m_j \in [n]$, the event that leads to the final global tree as a chain is

$$\begin{aligned} E_t &= \mathbb{1}(R_{1,m_2} < 1)\mathbb{1}(R_{1,m_3} < 2, R_{2,m_3} < 2)\ldots\mathbb{1}(R_{1,m_t} < t-1, R_{2,m_t} < t-1, \ldots, R_{t-1,m_t} < t-1) \quad (21)\\ &= \prod_{j=1}^{t} \mathbb{1}(E_{j,m_j}). \quad (22) \end{aligned}$$

Since we have assumed that $n \gg (\ell t)^2$, it follows from Theorem 4.1.1 that it suffices to compute the expression

$$\mathbb{E}[\prod_{j=1}^{t} \mathbb{1}(E_{j,m_j})|\text{all } \{m_j\}_{j=1}^{t} \text{ are distinct}] \tag{23}$$

Note that if all the m_j's are distinct, we have

$$\begin{aligned} &\mathbb{E}[\prod_{j=1}^{t} \mathbb{1}(E_{j,m_j})|\text{all } \{m_j\}_{j=1}^{t} \text{ are distinct}] && (24)\\ &= \prod_{j=1}^{t} \mathbb{E}[\mathbb{1}(E_{j,m_j})|\text{all } \{m_j\}_{j=1}^{t} \text{ are distinct}] && (25)\\ &= \prod_{j=2}^{t}\prod_{m=1}^{j-1}(1 - e^{-m/D}) && (26)\\ &= \prod_{j=1}^{t-1}(1 - e^{-j/D})^{t-j} && (27)\\ &\leq (1 - e^{-1/D})^{t-1}. && (28) \end{aligned}$$

Using the notation of Theorem 4.1.1, we have

$$F_t(D) = \prod_{j=1}^{t-1}(1 - e^{-j/D})^{t-j}. \tag{29}$$

We now claim that if $\ell \geq D\left(\ln t - \ln\ln\frac{1}{\delta}\right)$, we have $F_t(D/\ell) \geq \delta - o(1)$. Let $c = \ln\frac{1}{\delta}$.

Indeed, in this case, we have $e^{-\ell/D} \leq \frac{\ln \frac{1}{\delta}}{t}$. Hence,

$$\begin{aligned}
F_t(D/\ell) &\geq \prod_{j=1}^{t-1} \left(1 - \frac{c^j}{t^j}\right)^{t-j} && (30)\\
&= \left(1 - \frac{c}{t}\right)^{t\,\frac{t-1}{t}} \prod_{j=2}^{t-1} \left(1 - \frac{c^j}{t^j}\right)^{\frac{t^j}{c^j}\,\frac{c^j\,(t-1)}{t^j}} && (31)\\
&\geq e^{-c} - o(1) && (32)\\
&= \delta - o(1). && (33)
\end{aligned}$$

Conversely, we show that if $\ell \leq D\left(\ln t - \ln\ln \frac{1}{\delta}\right)$, then $F_t(D/\ell) \leq \delta + o(1)$.
Indeed, in this case we have $e^{-\ell/D} \geq \frac{\ln \frac{1}{\delta}}{t}$, and

$$\begin{aligned}
F_t(D/\ell) &\leq (1 - c/t)^{t-1} && (34)\\
&= e^{-c} + o(1) && (35)\\
&= \delta + o(1). && (36)
\end{aligned}$$

□

5 Finalization Layer

The last subsystem of Esperanza is finalization: a mechanism by which transactions can be clearly marked as final, or confirmed. Although the Esperanza proposal mechanism alone provides strong confirmation guarantees, the guarantees are ultimately probabilistic. This is unacceptable in many financial applications; hence the need for an explicit finalization protocol. Esperanza's finalization mechanism has two components: the first is a traditional, voting-based finalization protocol. In conjunction with this finalization protocol, we propose a network-aware approach to voting called Mahalo. Although Mahalo is compatible with a broad class of voting-based consensus mechanisms, we present it in the context of Casper the Friendly Finality Gadget (FFG), which was developed for Ethereum's transition from PoW to PoS [?]. Casper FFG is a voting protocol based on classical Byzantine fault-tolerant algorithms. For concreteness, we describe the mechanics of both Casper FFG and Mahalo in this section.

5.0.1 Casper FFG [?]

Finalization protocols are typically run for a specific block (rather than a transaction). In Casper FFG, every ηth block in the blockchain is designated a *checkpoint*. The finalization protocol is run on every checkpoint by a set of special nodes called *validators*. A node becomes a validator by depositing a minimum amount of tokens into a 'staking transaction', which locks up the validator's funds until she decides to no longer act as a validator. In principle, anyone can be a validator; however, the minimum deposit amount limits the set of nodes that can act as validators in practice. This large deposit is needed to provide security properties against Byzantine actors; we discuss these properties in Section ??.

A checkpoint can exist in one of three states: 1) null, 2) justified, or 3) finalized. The default state of a checkpoint is null, and only checkpoints in the 'finalized' state are considered finalized. To become finalized, the checkpoint must first pass through the justified state.

To explain the process of justifying and finalizing checkpoints, we introduce two terms: supermajority and vote. A *supermajority* denotes a pre-defined threshold for the weighted sum of stake deposited by each validator; by default, this threshold is typically set at 2/3. For example, if there are two validators, A who deposited 20 tokens and B who deposited 10 tokens, then A qualifies as a supermajority of validators because he deposited at least $\frac{2}{3}$ of the stake in the validator pool. In the remainder of this discussion, we will use the phrase 'supermajority of validators' when we actually mean 'supermajority of validators weighted by stake'.

Let h_s denote the *height* of a checkpoint, or the number of blocks between it and the genesis block. A *vote* is a cryptographically-signed data structure consisting of five fields:

1. A source checkpoint s
2. The source checkpoint's height h_s
3. A target checkpoint t
4. The target checkpoint's height h_t
5. The validator node's ID

Hence, we can think of each vote as linking two checkpoints of different heights in the blockchain; note that the target height must always be greater than the source height. For convenience, we let $v_i(s,t)$ denote a vote by validator i from source checkpoint s to target checkpoint t.

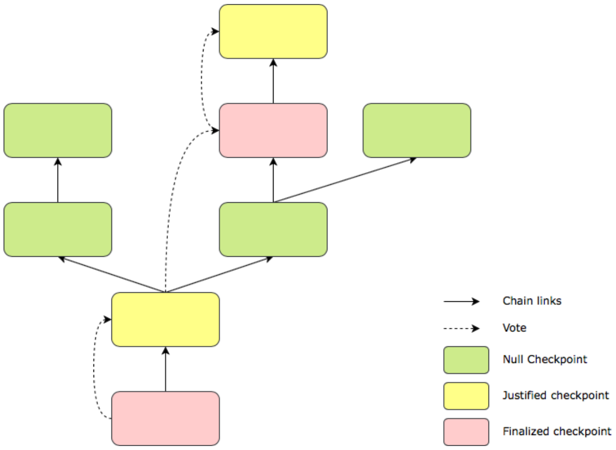

Figure 1: Example of justification and finalization in Casper FFG

If a validator i believes that target checkpoint t belongs in the blockchain that contains checkpoint s, it votes for that pair by broadcasting the vote message $v_i(s,t)$ to the rest of the network. A pair of checkpoints (s,t) is called a *supermajority link* if a supermajority of validators cast a vote for that pair. A checkpoint t is considered *justified* if a supermajority of validators have voted for (s,t) and s is either a justified or a finalized checkpoint (Figure 1). The genesis block is always considered finalized. A checkpoint s is considered *finalized* if s is justified and there exists a supermajority link (s,t) where $h_t = h_s + 1$. That is, t is a child of s.

In addition to these baseline rules, Casper FFG proposes two *slashing conditions*; if a validator is caught violating either of these slashing conditions, it loses its entire stake deposit. The conditions are:

1. Validator i casts two votes for a target at the same height. That is, it produces votes $v_i(s,t_1)$ and $v_i(s,t_2)$ such that $h_{t_1} = h_{t_2}$, and $t_1 \neq t_2$.
2. Validator i casts two votes $v_i(s_1,t_1)$ and $v_i(s_2,t_2)$ such that $h_{s_1} < h_{s_2} < h_{t_2} < h_{t_1}$.

These slashing conditions are the basis for Casper FFG's security guarantees, which are discussed in Section **??**. Slashing conditions are enforced by other nodes; if a node provides evidence of a validator violating the slashing conditions, the node receives a portion of the slashed funds.

5.0.2 Mahalo: Network-aware voting

Casper FFG is based on practical Byzantine fault tolerance (PBFT), a Byzantine fault-tolerant consensus protocol published in 1999 [**?**]. PBFT (and hence, Casper FFG) was originally designed to provide two key properties assuming no more than $\frac{1}{3}$ of nodes are Byzantine (i.e. $\beta < \frac{1}{3}$): safety and liveness. Roughly, safety means that the protocol is correct, whereas liveness means that the protocol eventually concludes. Neither safety nor liveness guarantees anything about a consensus protocol's efficiency, either in terms of time to consensus or communication cost. In particular, PBFT was originally designed for a partially synchronous communication model in which communications are assumed to arrive within some maximum network delay D [?]. In many ways, this is a worst-case assumption; average Internet packet delays can be substantially smaller than worst-case delays, which must contend with congestion, hardware failures, and other atypical events. Nonetheless, PBFT must accommodate worst-case assumptions in order to guarantee both safety and liveness.

We maintain that Casper FFG does not require the same robustness properties as PBFT, due mainly to the unique sequential nature of blockchains. In particular, if consensus (finalization) is not reached at one checkpoint, Casper FFG gracefully moves on to the next checkpoint. When any checkpoint is eventually finalized, it implicitly finalizes all the checkpoints before it. Thus, lack of consensus at a particular checkpoint height is not a catastrophic event, as it might be in other applications. Because of this, we can relax some of PBFT's worst-case assumptions about network performance in order to reduce delay and increase throughput.

Mahalo is a modification to the voting protocols of Casper FFG that specifies how and when validators vote for a particular link in the blockchain. [?] does not specify when validators should vote for a particular link; this decision is left to the validator node implementation. A common default implementation choice is for validators to vote as soon as they receive a checkpoint. We call this approach the *immediate vote* algorithm.

The idea of Mahalo is simple (pseudocode in Algorithm 1): upon receiving a new checkpoint at height h, a validator starts a random, exponentially-distributed timer. Once the timer ticks, the validator tallies all vote messages received from other nodes with a target checkpoint at height h, and computes the plurality checkpoint with the most votes. If the validator has already received the plurality checkpoint (as opposed to just receiving votes for it), then the validator votes for the plurality checkpoint. Otherwise, it waits until receiving the plurality checkpoint, while dynamically updating its estimate of the plurality checkpoint. In case of ties, the validator picks a checkpoint randomly. This algorithm bears conceptual similarities to random backoff in the ALOHA protocol for medium access control; hence the name.

Perhaps surprisingly, Mahalo reduces the time to consensus by increasing the time required for each individual validator to vote. The intuition is that by obtaining more complete information about the state of the network, Mahalo produces better-informed validators, which reduces the likelihood of forking. We discuss these properties in detail in Section **??**.

Algorithm 1 Mahalo

Validator i has not voted for a checkpoint at height h yet height h rxCheckpts$[h] = \{\}$ checkpoint t rxVotes$[h,t] = 0$ true i receives checkpoint t with height h rxCheckpts$[h]$ $+=$ $\{t\}$ voteTime = CurrentTime() + exponentialRand(λ) CurrentTime() $<$ voteTime i receives vote $v_j(s,t)$ where $h_t = h$ rxVotes$[h,t]$ $+= 1$ pluralityCheckpt $= \arg\max_t\{\text{rxVotes}[h,t]\}$ pluralityCheckpt $\notin$ rxCheckpts$[h]$ continue tallying votes and updating rxCheckpts choose a justified checkpoint s and cast vote $v_i(s, \text{pluralityCheckpt})$

5.1 Efficiency Properties

Mahalo is designed precisely to improve the efficiency of the finalization process. In particular, although we present it in the context of Casper FFG, it can be added to *any* voting-based finalization protocol. In this section, we discuss simulations evaluating the relative benefits of Mahalo over vanilla Casper FFG.

Our simulator is a fork of Charles Bournhonesque's and Olivier Moindrot's Casper FFG simulator.[1] It consists of 1,200 lines of Python, and simulates a network of validators running Casper FFG + Mahalo [2]. The continuous-time simulator assumes that all nodes have equal stake in the finalization process, and hence are weighted equally in the voting tally. Every $T_B = 3.5$ seconds, a proposer is randomly selected from the nodes; the proposer appends a new block it to the head of the its local tree, and broadcasts the new block across the network. We model the end-to-end propagation delay between any two nodes as exponential with mean D, in accordance with the measurement results of [?]. Every $T_C = 175$ seconds (i.e. every $T_C/T_B = 50$ blocks), we have a *checkpoint*, on which Casper FFG is run. The quantity T_C/D is closely tied to the performance gains of Mahalo; our simulations suggest that Mahalo's efficiency gains are particularly pronounced when T_C/D is close to one, i.e., when checkpoint times and propagation delays are similar.

[1] `https://github.com/ethereum/casper`

[2] Code at `https://github.com/ranvirranaiitb/pos`

Intuitively, this regime exhibits high levels of forking because proposers are unlikely to have the most recent blocks; hence, collecting more information during the voting process helps reduce forking and confirmation times. For simplicity, we use the same network model for votes and blocks, even though votes are smaller than blocks, and therefore likely to propagate faster. The experiment is run for a total of T seconds.

In cryptocurrencies, if a block does not end up in the main chain due to forking, the transactions contained in that block are added to a new block by proposers. We model this by assuming that once a block is excluded from the main chain, it is immediately re-proposed. For simplicity, we assume that all blocks in a single epoch contain distinct transactions. Furthermore, we assume that a block (or transaction) gets finalized in each epoch with equal probability, independently across transactions. These assumptions are important for implicitly measuring one of our performance metrics, finalization delay.

Metrics We measure two key performance metrics: transaction finalization delay and throughput. Let F_B denote the event that block B is eventually finalized, meaning that the block or one of its descendant checkpoints is eventually finalized. Let τ_B denote the (random) time until a given block B is finalized, conditioned on event F_B (i.e., conditioned on B being finalized). *Transaction confirmation delay* τ is defined as

$$\tau := \frac{[\tau_B]}{\P(F_B)}$$

We also measure *throughput*, which is defined as the number of finalized blocks per unit time:

$$\lambda := \frac{|\{B : \tau_B \leq T\}|}{T}$$

Results To evaluate Mahalo, we simulated the effects of the average wait time, for an average delay $D = 7.5$ s, a block time of 3.5 s, and a total runtime of $T =??$ seconds. Figure 2 illustrates the resulting confirmation latency as a function of the wait time. The expected confirmation delay appears to be minimized when the wait time is equal to the network latency, regardless of the number of validators in the system. Notably, Mahalo did not lead to statistically-significant changes in throughput.

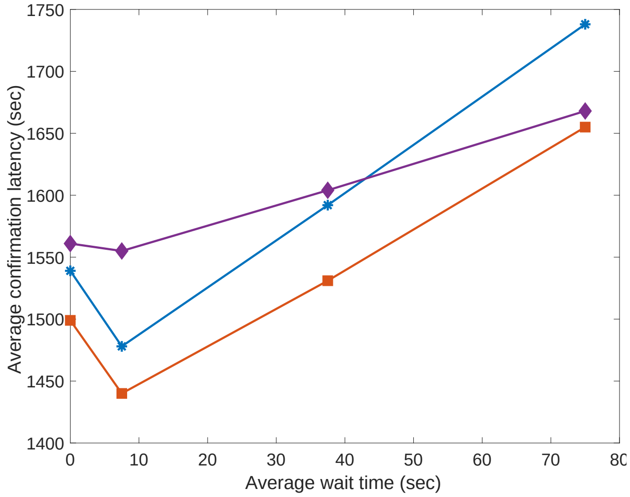

Figure 2: Expected confirmation latency as a function of expected wait time in Mahalo.