Wavelet: A decentralized, asynchronous, general-purpose proof-of-stake ledger that scales against powerful, adaptive adversaries

Kenta Iwasaki
kenta@perlin.net

Heyang Zhou
heyang.zhou@perlin.net

May 26, 2019

Abstract

Wavelet introduces a novel family of directed-acyclic-graph (DAG)-based consensus protocols. It is designed to alleviate the numerous scalability dilemmas predicated in decentralized ledgers, such as those that utilize either the longest chain rule, or some variant of stake delegation or committee election scheme. Wavelet guarantees irreversibility of transactions, and a consistent total ordering of transactions without any compromise on safety, performance or liveness; enabling features such as transaction graph pruning and Turing-complete smart contract execution. The safety and liveness of Wavelet is solely dependent on the safety and liveness of any arbitrarily chosen Byzantine fault-tolerant binary consensus protocol executed within the Wavelet framework. Unlike prior works, Wavelet requires minimal configuration of a minuscule set of system parameters to work in a large range of practical network settings where communication is only partially synchronous.

1 Introduction

All distributed systems suffer from what is known as the *scalability dilemma*. The dilemma forces system designers to have to make tradeoffs between performance, consistency and availability; aspects which are of utmost importance when it comes towards the creation of practical, resilient, fault-tolerant applications. Least to say, a particular class of distributed systems, blockchains, suffer from the aforementioned dilemma.

Below, we perform a trivial analysis over results from classical computer science literature to describe and analyze a plethora of faults that render numerous modern blockchain constructions impractical.

1.1 The scalability dilemma

Recall the multiple readers, multiple writers problem [4]. There exists n processors all trying to concurrently read and write to some shared resource R. For each and every additional processor alloted to the system, there requires an exponential increase in the amounts of coordination among all n processors such that they may read or write to resource R.

Now, reposition this problem into a distributed setting where there exists communication latency between each and every processor. An equivalence may be established to this problem: it is fundamentally difficult to efficiently coordinate a network of nodes to maintain a distributed ledger without introducing exponential communication costs.

Cumbersomely, accounting for faulty processors in designing a protocol to coordinate multiple readers and multiple writers to interact with resource R introduces significant additional overhead. Throughout this paper, we denote this problem as the *scalability dilemma*.

1.2 The longest chain rule does not scale

Consider a network of n nodes. A difficulty system, mechanized as a decentralized random lottery amongst all n nodes is established. Within a short time interval Δ, a subset F of n nodes may propose a block of transactions to append to the longest weighted path of a directed-acyclic graph of blocks G shared across the entire network.

We denote this distributed system to follow the *longest chain rule* [10]. Bitcoin, Ethereum, and several other decentralized ledger architectures are noteworthy examples of distributed systems that follow such a rule.

Within the time interval Δ, there may exist at most $|F| - 1$ conflicting blocks attempting to be appended to the longest weighted path of G, otherwise known as the longest chain. Resolving conflicts between the $|F| - 1$ blocks requires nodes in F to append additional blocks to the blocks that they have proposed. The difficulty system makes this task for each individual node $\in F$ require exponential time complexity to be able to resolve conflicts, which is arduous.

A relaxation of this scenario is that the longest chain rule is effectively trying to have n nodes compete in having the chance to read and write to some resource R, being the longest chain, which is simply not scalable for large n. Therefore, any distributed system that follows the longest chain rule suffers from the scalability dilemma.

1.3 Committees improve performance by weakening security

Consider a network of n nodes, with C being a subset of the n nodes, denoted as a committee. The committee is responsible for verifying and processing all transactions within the network.

The methodology in electing the committee within whatever time interval permits does not matter; whether it be through means of stake delegation [9], verifiable random functions [6], or through decentralized electoral voting schemes [2]. Noteworthy examples of distributed ledgers that utilize committees are Algorand, Tendermint, Ouroboros Praos, Dfinity, and EOS.

Such a scheme may scale variably, given that only communication between all nodes $\in C$ is necessary to maintain a distributed ledger. However, should $|C|$ be small, any form of denial-of-service attack or collusion within C would jeopardize the entire network.

To minimize risks of such attacks from happening would require scaling up the size of C. This contradicts however the benefits of maintaining a small committee given that a larger committee exponentially increases communication complexity. Therefore, any distributed system that relies on a subset of nodes to verify and process all transactions for its entire network suffers from the scalability dilemma.

1.4 A partial ordering of transactions has limited practicality

Notably, several consensus protocols that orient consensus over the replication of a directed-acyclic-graph of transactions across nodes, such as Avalanche [12], are:

1. only able to guarantee a partial ordering of transactions,
2. unable to guarantee irreversibility of transactions, and
3. susceptible to numerous challenges when seeking to enable new nodes to bootstrap and join the network.

The inability to fulfill these guarantees restricts a ledger in being unable to provide a wide plethora of features. For instance, a lack of guarantee for a total ordering of transactions would render a ledger incapable of being anything more than a payments platform.

It is impossible for a Turing-complete smart contracts platform to be built on top of a ledger that only

guarantees a partial ordering of transactions. This is because the sequence in which smart contracts are executed would be totally different across different nodes in the network.

1.5 Network assumptions and protocol guarantees

Definition 1.1. Adversarial model. We assume a powerful, adaptive adversary capable of observing the internal state of every node within the network. The adversary may uniformly drop, reorder, or redirect any amount of messages sent from each node, so long as $1/k$ messages successfully gets delivered by each node to their intended recipient with k being some small, fixed security parameter.

Wavelet guarantees both strong safety and liveness assuming that all communication in the network is partially synchronous [5] with soft probabilistic bounds with respect to k given the prescribed adversarial model. Under these conditions, Wavelet is able to achieve an irreversible and consistent total ordering and application of all virtuous transactions within the network.

2 The Wavelet Protocol

The protocol comprises of two processes running in parallel: *gossiping*, and *querying*.

Definition 2.1. Identity. Every node in the network assumes a public-key infrastructure (PKI), in which nodes are identified by their associated public keys.

Definition 2.2. Hash. Denote a cryptographic hash function H which is random oracle in-differentiable. We refer to bit-length of all outputs of the hash function H as $|H|$.

Definition 2.3. Account. Accounts are referenced by an associated public key, and comprise of both a balance **balance** of virtual currency alongside a nonce. The nonce **nonce** is associated as a counter that sequentially counts changes made to a specified account as a procedure for preventing replay attacks.

Definition 2.4. State. Every node in the network maintains a ledger state S, representative of an immutable, Merkelized key-value store that may produce a Merkle root δ.

Definition 2.5. Graph. Every node in the network maintains a directed-acyclic-graph G whose vertices comprise of either accepted or pending transactions, and whose edges are representative of a causal topological ordering.

Definition 2.6. Transaction. A transaction tx denotes an atomic unit of changes to be applied locally by a node to their state S from the perspective of its designated creator. Each and every transaction may be referenced by its unique ID, which is the checksum of all of its contents and metadata. Transactions are vertices of G that casually reference a set of parent vertices tx.parents by their ID. Transactions reference their depth with respect to graph G as tx.depth. A transaction denotes its proposal of changes to apply to some state S through its payload tx.payload, which is used to invoke some unique operation designated under tx.tag. More compactly, a transaction $\text{tx} = (\text{id}, \text{depth}, \text{parents}, \text{tag}, \text{payload})$.

Definition 2.7. Transaction authorship. A transaction tx applies its changes on state S with respect to its creator. The creator of the transaction is referenced by their public key as tx.creator. It is optional, though possible, for the creator of the transaction to delegate the responsibility of proposing and relaying the contents of their transaction tx to the network towards another node, referenced by their public key as tx.sender. A transaction must always specify both its creator and sender, even if the transaction's sender should be the creator themselves. The creator of the transaction signs and appends the signature of **nonce**$\|$tx.tag$\|$tx.payload to tx. The sender of the transaction signs and appends the signature of the entire contents of tx, excluding its unique ID.

2.1 Creating a transaction

Assume two parties, Alice and Bob. Alice holds a large amount of virtual currency, and wants to create a transaction tx that deducts money from her account, and adds money to Bob's account. Alice pries at her view of the graph G, and selects parents for transaction tx to be at most MAX_NUM_PARENTS of all available transactions from the last DEPTH_DIFF depths of G's frontier that do not reference any children transactions (such that they are leaf nodes of graph G).

Definition 2.8. Transaction depth. After selecting the parents for transaction tx, the depth of the transaction, tx.depth, is computed and appended to tx as:

$$\text{tx.depth} = \max\{\text{parent.depth} \mid \text{parent} \in \text{tx.parents}\} + 1 \tag{1}$$

Definition 2.9. Transaction seed. After the depth of the transaction is computed, we denote a unique checksum, otherwise known as the seed of the transaction, tx.seed, to be computed to be:

$$\text{tx.seed} = H(\text{tx.sender} \| \text{tx.parents}) \tag{2}$$

It is important to remark that both the ID and the seed of the transaction tx are simply stored in-memory, and may be trivially computed by any other node in the network. Upon signing transaction tx, Alice excludes both the ID and the seed of the transaction before producing and appending both creator's and sender's signatures to tx.

2.2 Gossiping

Alice then signs, and proceeds to publish her transaction tx out to the network using any arbitrary eventually-consistent asynchronous gossiping protocol $\mathcal{P}_{\text{gossip}}$. After some eventual period of time Δ, should there exist no publication of new transactions, all nodes would eventually have the contents of their graph G to be consistent.

All transactions that are newly created, relayed, or otherwise received by this gossiping process have yet to be fully accepted by the network. The importance of the gossiping process is to ensure that nodes eventually construct an equivalent and consistent view of their graphs G.

An assumption is made that an overlay network protocol $\mathcal{P}_{\text{overlay}}$ is enforced to allow for nodes to send and eventually receive all gossiped transactions under the presence of an adversary that is capable of dropping, reordering, or indefinitely delaying a large percentage of all messages sent by individual nodes.

Definition 2.10. Depth assertion. Upon receipt of a transaction tx from another node, Alice performs an assertion before appending tx to her view of the graph G that the depth of the parents of the transaction with respect to tx.depth do not differ by more than DEPTH_DIFF. The fixed system parameter DEPTH_DIFF restricts transactions from being able to select parent transactions from any depth within graph G.

Definition 2.11. Order assertion. Upon receipt of a transaction tx from another node, Alice performs an assertion before appending tx to her view of the graph G that the parents of the transaction tx.parents are lexicographically sorted in ascending order.

Definition 2.12. Parent selection limit. Upon receipt of a transaction tx from another node, Alice performs an assertion that tx has at most MAX_NUM_PARENTS parents referenced before appending tx to her view of graph G.

2.3 Querying

Apart from the gossiping process, another process runs in parallel at each node that is responsible for ordering, finalizing and applying transactions to each and every nodes' local state S.

The querying process runs incrementally as the graph G gets built up on each node; finalizing batches

of transactions from the root of G to the frontier of G. More specifically, the querying process' objective is to divide the graph G up consistently on each node into *consensus rounds*.

Definition 2.13. Consensus round. A consensus round R_r is a non-overlapping depth interval, notated as $R_r = (r, \text{tx}_{\text{start}}.\text{depth}, \text{tx}_{\text{end}}.\text{depth})$, where $r \in [0, \infty)$, and tx_{start} and tx_{end} are the starting and ending point transactions of the depth interval the consensus round comprises of. The starting and ending points of a consensus round may only be transactions that are marked as *critical*.

Definition 2.14. Critical transaction. A transaction tx is marked to be critical only if the number of prefixed zero bits of its seed $\text{tx.seed} \geq D_r$, where D_r is a dynamic system parameter, otherwise known as a difficulty parameter, that is unique with respect to some given consensus round r.

Assume that Alice is on the very first consensus round R_0. As the gossiping process progresses and builds up G, upon the discovery of a critical transaction, Alice will propose a candidate ending point for her current round R_0, which will be the starting point for the next round R_1. The proposal is that $R_1 = (1, \text{root.depth}, \text{tx}_c.\text{depth})$ where root is the root transaction of G, and tx_c is the very first critical transaction spotted by Alice. Alternatively, apart from Alice proposing the ending point for round R_0, Alice may also learn from her peers of the existence of alternative ending points for R_0.

Alice then proceeds to execute any arbitrary Byzantine fault-tolerant binary consensus protocol $\mathcal{P}_{\text{query}}$ to garner an unbiased opinion from the network over the finalized contents of consensus round R_1.

Upon finality over the contents of consensus round R_1, a breadth-first traversal is performed to find all transactions in all paths of G whose starting point is root, and whose ending point is $\text{tx}_{c'}$ where $\text{tx}_{c'}$ is the finalized starting point of R_1, and finalized ending point of R_0.

The order in which the breadth-first traversal is performed is the finalized ordering of all transactions to be applied locally to each and every nodes state S for round R_0. The subset of all transactions traversed in the finalized consensus round R_0 are marked as accepted, with all other transactions within the depth interval of R_0 marked to have failed.

Definition 2.15. State assertion. Denote all existing consensus rounds R with equivalent indices r at some point in time, appended with Merkle roots δ which are computed by applying all accepted transactions of some $R_i \in R$ to some state S. Upon the finality of a single consensus round $R_i \in R$, nodes may apply accepted transactions of R_i to their state S to recover a Merkle root δ'. Nodes are able to then assert the integrity of their newly updated state S' with respect to the finalized round R_i by asserting that $\delta' = \delta$.

All accepted transactions are then applied to each and every nodes state S, with each application incrementing the **nonce** of the transactions creator. The nodes then repeat the entire querying process from then on by searching for the next ending point critical transaction for consensus round R_2, and set the root of consensus round R_1 to be $\text{tx}_{c'}$. This querying process inductively proceeds forward for the next consensus round and onwards.

2.4 Partial synchrony

A crucial aspect of the protocol is computing an ideal value for the dynamically changing system parameter D_r given some consensus round r. To paint a picture, recall that gossiping is a process that results in graph G eventually becoming consistent across all nodes in the network. Hence, for larger n where n is the number of nodes in the network, the larger the time upper bound Δ is in having gossiping converge into completing its target objective of having all nodes have consistent graphs G.

Larger n also implies that a larger amount of transactions may be proposed per second, as there exists more sources that may spawn and distribute transactions throughout the network. Denote the upper bound frequency in which a single node N_i is able to create and gossip transactions per second as f. The maximum number of transactions that may be created at any instance of time is thus nf, which is also representative of the max possible load the network may be under at any moment in time.

Denote the time it takes to finalize a single consensus round to be t', and the number of transactions that encompassed by the non-overlapping depth interval representative of a single consensus round given some difficulty parameter D_r to be f'.

Conjecture 2.1. Partial synchrony bounds. Denote the total number of transactions pending to be accepted in G to be $|G_{\text{pending}}|$, and the total number of transactions finalized in G to be $|G_{\text{finalized}}| = |G| - |G_{\text{pending}}|$ at some given time t. A transaction in G is considered to be finalized if its depth is less than or equal to the depth of the ending point transaction of the most recently finalized consensus round a node is aware of. If $nf > \frac{f'}{t'}$, then $\lim_{t\to\infty} \frac{d}{dt}(|G_{\text{pending}}| - |G_{\text{finalized}}|) > 0$. Conversely, if $nf \leq \frac{f'}{t'}$, then $\lim_{t\to\infty} \frac{d}{dt}(|G_{\text{pending}}| - |G_{\text{finalized}}|) = 0$.

As a consequence of Conjecture 2.1, should consensus rounds not sufficiently finalize large enough depth intervals of G over time, $|G_{\text{pending}}|$ would be *strictly increasing* over time with respect to $|G_{\text{finalized}}|$. This represents a consequence of the protocol exhibiting a liveness fault due to misconfiguration of difficulty parameter D_r, where consensus rounds are not being finalized fast enough such that there exists too many pending transactions in G.

To avoid such a liveness fault, we must adjust the difficulty parameter D_r in correspondence to the frequency in which transactions are created within the network at any given moment in time. To do so requires a verifiable, deterministic mechanism that allows nodes to independently compute an approximation of the total load of the network to adjust the interval span of each and every newly finalized consensus round.

We propose a novel method to estimate the network load as a factor that may then be logarithmically composited to produce a sufficient D_r which dynamically attunes the network to prevent liveness faults.

2.5 Approximating network load

Recall the parent selection algorithm denoted in Section 2.1, and denote two honest nodes N_1 and N_2. Consider the two nodes N_1 and N_2 out of the entire network of $|N|$ nodes to be individually creating and gossiping nearly, or otherwise at most, f transactions per second.

Assuming that N_1 and N_2 at any given time t have similarly composited graphs G resultant of gossiping, the creation of arbitrary transactions from both N_1 and N_2 at any point in time t must comprise of similarly selected sets of parent transactions.

Denote a time interval Δ where strictly only nodes N_1 and N_2 are creating and gossiping at most $2f$ transactions per second. The maximum number of transactions that may coexist within newly created graph depths by the transactions created by nodes N_1 and N_2 throughout the time interval Δ is 2.

Inductively:

Conjecture 2.2. Network load phenomenon. Denote a time interval Δ where i honest nodes are creating and gossiping nearly, or otherwise at most, if transactions per second. The maximum number of transactions that may coexist within newly created graph depths constructed throughout the time interval Δ is i.

Taking advantage of this phenomenon allows us to create a deterministic, verifiable heuristic that allows for nodes to approximate the load or congestion of its residing network effectively.

Definition 2.16. Network load factor. Assuming Conjecture 2.2, denote the ancestor set of a transaction to be tx.ancestors given some transaction tx. Given a consensus round $R_r = (r, \text{tx}_{\text{start}}.\text{depth}, \text{tx}_{\text{end}}.\text{depth})$, we approximate a load factor L_r depicting the congestion of all nodes within a network N for round r to be:

$$L_r = \frac{|\text{tx}_{\text{end}}.\text{ancestors}|}{\text{tx}_{\text{end}}.\text{depth} - \text{tx}_{\text{start}}.\text{depth}} \tag{3}$$

Theorem 2.1. Adversarial resistance. Assuming Conjecture 2.2, denote an adversary A that creates transactions at arbitrary depths within the depth interval represented by some consensus round R_r. It is impossible for the adversary A to have load factor L_r decrease, or otherwise increase by more than 1 with a single valid transaction whose parents are arbitrarily chosen.

Proof. The creation of some arbitrary transaction tx' by adversary A may only either increase the height of the graph G by 1, or otherwise increase the number of transactions at depth tx'.depth of graph G by 1 where tx'.depth $\leq$ tx_{end}.depth. For the former case, both $|\text{tx}_{\text{end}}.\text{ancestors}|$ and tx_{end}.depth would trivially increase by exactly 1. This leaves the latter case allowing for arbitrary increments of L_r, as a result of widening graph G. However, recall the parent selection algorithm from Definition 2.1: a honest node would only build transactions which reference parents that are leaf nodes such that they follow the former case. Hence, given a network with at least one honest node and some adversary A, the honest node would only build transactions that reference transactions made by adversary A that do not follow the latter case. Therefore, it is impossible for load factor L_r to increase by more than 1 with a single transaction, with both former and latter cases not allowing for any arbitrary decrement of L_r. □

2.6 Adjusting difficulty

Recall the condition which marks a transaction to be critical in Definition 2.14. Trivially, linearly adjusting the value of the system parameter D_r either exponentially increases or decreases the frequency in which a critical transaction may be created. The probability that a critical transaction is created given that the bit-length of a transactions seed is $|H|$ is thus $\frac{1}{2^{D_r}}$ where $D_r \leq |H|$.

To linearly adjust the frequency in which a critical transaction is created, we logarithmically scale the load factor L_r by two system parameters MIN_DIFFICULTY and DIFFICULTY_SCALE such that:

$$D_r = \text{MIN\_DIFFICULTY} + \text{DIFFICULTY\_SCALE} * \log_2 L_r \tag{4}$$

Logarithmically scaling L_r allows for linear amendments to the frequency in which a critical transaction is created given that D_r's growth rate is inversely proportional to that of $\frac{1}{2^{D_r}}$.

To figure out ideal fixed values for MIN_DIFFICULTY and DIFFICULTY_SCALE, recall that the lower communication latency is within a network, the larger the networks maximum load capacity is at some point in time. Denote an ideal network environment where there exists zero communication latency. We attune parameters through any optimization methodology to such an environment such that th protocol exhibits5 minimal liveness faults.

Should this newly derived parameter set allow for a network where there exists zero communication latency to follow the protocol with minimal liveness faults, these parameters may safely be applied to the execution of the protocol in more practical network settings where there exists non-zero communication latency.

This is because practical networks in comparison to an ideal zero-latency network are incapable of exhibiting more load than that of a zero-latency network. Thus, the parameters are properly attuned to have the protocol execute in a worst-case scenario environment.

3 Design

Definition 3.1. Wavelet. Wavelet is a family of consensus protocols, with individual variants of Wavelet denoted by the tuple $W = (\mathcal{P}_{\text{overlay}}, \mathcal{P}_{\text{gossip}}, \mathcal{P}_{\text{query}})$.

As a first practical step, we consider a Wavelet variant W where $\mathcal{P}_{\text{gossip}}$ is the S/Kademlia overlay network protocol [1], $\mathcal{P}_{\text{gossip}}$ is a barebones gossiping protocol where nodes spread information through all peers recorded in their individually maintained Kademlia routing tables, and $\mathcal{P}_{\text{query}}$ is the Snowball consensus protocol [12].

3.1 S/Kademlia

The choice of the S/Kademlia overlay network protocol is due to its formation of a De-Bruijn networking topology that allows for messages to be efficiently sent to a designated recipient with $O(\log n)$ hops where n is the number of nodes in the network.

S/Kademlia's static and dynamic cryptographic puzzles enforce that node identities are uniformly dispersed. This enforces that each and every node's individually maintained Kademlia routing tables are uniformly partitioned to allow for communication across the honest subset of the network to persist in spite of the adversarial model defined in Definition 1.1 dropping, redirecting, and delaying the delivery of messages.

Given the uniformity of node identities, a simple gossiping protocol may additionally be constructed on top of S/Kademlia where nodes simply gossip information across their neighbors inscribed in their individual routing tables. Assuming users are aware of at least one honest bootstrapping node, new users may easily join membership of the network.

3.2 Snowball

Snowball is a consensus protocol inspired by the Snowball sampling technique [7] that guarantees strong safety and liveness in a partially synchronous network where there exists an adversary whose behavior is defined in Definition 1.1.

Lemma 3.1. Strong liveness. The Snowball loop will eventually terminate given a set of n processors that are t-resilient, where t is the tolerable number of faults each individual processor may handle. There is a single case for termination of the Snowball loop, in which we successfully sample a color consecutively from k peers β number of times. Given the assumption that at most 1/3rd of all processors are faulty, we reformulate the rigidness of Snowball's liveness as the following:

Denote a random variable X with an arbitrary discrete distribution whose sample space comprises of two elements: **R**, and **B** (the colors), and whose probabilities do not comprise of zero measure. Prove that infinitely sampling from X would eventually yield a sequence of β of the exact same element.

Proof. Assume that it is impossible for the Snowball loop to ever terminate given a network of n nodes that are t-resilient. However, recall the results of the *infinite monkey* theorem [8]: given an infinite sequence of infinite strings, where each character of each is string is chosen uniformly at random, any given finite string almost surely occurs as a prefix of one of these strings.

We may relax the assumption that characters sampled from X follow a uniform distribution, so long as probability measures are non-zero, and each sampling event is independent based on the converse result of the second Borel-Cantelli lemma [3].

This thus poses a contradiction: it is almost surely true that infinitely sampling from X would yield a string of β length that only comprises of the exact same element. Therefore, assuming that communication within the network is partially synchronous, it must be the case that the Snowball loop will eventually terminate. □

Lemma 3.2. Strong safety. Upon termination of the Snowball loop for all n processors, it is guaranteed that they all agree on the value of some preferred color C.

Proof. Following Lemma 3.1, assume that the preferred color C is not consistent across all nodes. However, trivially, the preferred color C upon termination of the Snowball loop has the maximum count over all other colors. This poses a contradiction: it must be the case that the preferred color C is consistent across all nodes. □

Following Lemmas 3.1 and 3.2, it must be the case that Snowball is able to achieve binary consensus with strong safety and liveness over a partially synchronous network of n nodes.

We emphasize however that the results from Lemma 3.1 does *not* prove that Snowball achieves strong liveness within a small, practical number of iterations under the denoted adversarial model. Nonetheless, given that Snowball inherently is a sampling mechanism, a wide variety of strategies may be bootstrapped for Snowball to converge to a consensus in a small, finite number of iterations.

Examples of such strategies include weighing a node's response based on the staking of virtual currency detailed below, alongside dynamically attuning the α parameter under the presence of adversaries [11].

3.3 Proof of stake

Nodes may stake amounts of virtual currency in order to variably garner more weight in assisting with the settlement of a consensus round. Such a strategy is trivially resistant to safety faults because the stake of an independent node is consistent across rounds. Having such a mechanism in place establishes resilience against Sybil and Eclipse attacks for the protocol.

Denote the discrete random variable $R_k \sim \mathrm{B}(k, p)$ to model the responses of each randomly sampled peer out of k peers as a set of k success probabilities p, and the discrete random variable S_k to model the distribution of stakes made by each of the k peers within the network. Given the product distribution $R' = S_k R_k$, we declare a sampling for a color to be successful should $\mathbf{E}[R' \geq \alpha]$ occur.

3.4 Transaction fees and rewards

A node with more weight in assisting with the settlement of a consensus round, otherwise known as a *validator*, naturally attracts the attention of both honest and adversarial nodes. In compensation for such circumstances, fees are deducted for every accepted transaction in a settled consensus round, and are dispersed to validators that assisted with the development of such a consensus round.

Numerous problems arise however: how do we rightfully match up a validators efforts in the settlement of a consensus round with some amount of reward? To address such an issue, an incentive system is devised that rewards validators based on their stake and activity in appending new transactions to graph G that references as many ancestries as possible through a verifiably random voting scheme.

To illustrate how this system works, denote a transaction tx that has just been finalized. Then:

1. Let eligible = {tx'.sender $\neq$ tx.sender | tx' $\in$ tx.ancestors}. If |eligible| = 0, terminate the procedure.
2. Let validators = {tx'.sender | tx' $\in$ eligible}. Concatenate and hash together all of {tx'.id | tx' $\in$ eligible} using H to construct a checksum that acts as an entropy source E.
3. Let X' be some threshold value that is the last 16 bits of E, and denote a random variable X which models a weighted uniform distribution over the set of all candidate validators. We then select a validator $v \in$ validators that has a weight $X \geq X'$ as the transaction fee reward recipient.
4. Transfer some amount of transaction fees from tx.sender to the intended reward recipient v.

Such a system incentivizes validators to actively create and broadcast new transactions with a large selection of ancestries to graph G. Non-trivially, the system does not suffer from the *nothing-at-stake* problem given that any effort to attain rewards requires that a validator must assist with the validation of large, exponentially growing amounts of transactions.

The reward distribution scheme combined with the S/Kademlia overlay protocol $\mathcal{P}_{\text{overlay}}$ naturally defers the *rich gets richer, poor gets poorer* dilemma favorably as well, since the network topology does not discriminate such that peers must interact with high-stake validators.

4 Discussions

We discuss a few key concerns to consider with the proposed protocol design described in Section 3, assuming that communication across the network is partially synchronous.

Theorem 4.1. Fork resilience. We consider two nodes to *fork*, or otherwise *diverge* if the two nodes' view of the contents of some round R_i that is finalized at some point in time are not equivalent. Given two honest nodes N_1 and N_2 in a network with a tolerable number of adversarial nodes, it is impossible for nodes N_1 and N_2 to fork.

Proof. Recall the results from Theorem 2.1: honest nodes would only build transactions that refrain from referencing parent transactions that intentionally increases the number of transactions at some depth d. The critical transaction proposed by honest nodes to end round R_i would therefore only reference parent transactions created by other honest nodes that do not widen the graph G, with the converse being that critical transactions proposed by adversarial nodes to end round R_i would not be accepted by honest nodes. Therefore, it is impossible for adversarial nodes to build transactions forking graphs G maintained by honest nodes that are referenced by the ancestry of the ending transaction for round R_i.

Now, to consider fork resiliency under the activity of honest nodes, assume N_1 and N_2's graphs G are not equivalent, and either one of them proposes a critical transaction to end round R_i. Denote the node that did not propose the critical transaction to end round R_i to be N_1', and the other node to be N_2'. Recall Definition 2.15: N_1' would fail to reconstruct the Merkle root for the critical transaction proposed by node N_2' given non-equivalent graphs. Only once N_1' and N_2''s graphs G are similar may N_1' vote for N_2''s proposed critical transaction to end round R_i under the Snowball consensus protocol. Hence, for two nodes to fork would require for there to be a violation in the safety property of the Snowball consensus protocol.

Therefore, as it is not possible for the contents of round R_i to diverge for honest nodes given the creation of well-received transactions by arbitrary nodes, it is impossible for two nodes N_1 and N_2 in a network with a tolerable number of adversarial nodes to fork. □

As a consequence of the results of Theorem 4.1 however, it is possible that virtuous transactions by honest nodes may not be finalized under the round they were created within. This would result in transactions that are then rejected to have to be re-signed and re-submitted to the network with newly picked parent transactions, which is cumbersome.

It is left for future work for describing a solution to make the protocol more resilient against such an issue. Network-wide incentives over the construction of graph G, such as the expense of either virtual currency or computational resources, are viable approaches to tackling this issue.

5 Results

The Wavelet variant W described in Section 3 establishes the following guarantees under a partially synchronous network:

Definition 5.1. Total ordering of transactions. Any two nodes with individually maintained graphs finalize and apply the same set of accepted transactions in the exact same order.

Definition 5.2. Transaction irreversibility. All transactions whose depths are equal to or less than any finalized consensus rounds ending points depth are marked to be accepted and finalized, and may not have its acceptance or finalization revoked under protocol.

These guarantees lend the Wavelet variant W to a few interesting prospects and features.

Definition 5.3. Succinct. All finalized transactions may safely be pruned, as finalized transactions are irreversible under protocol. Given that a payment transaction is in the order of a few hundred bytes, the

entire graph of transactions each node maintains may be stored *entirely* in-memory. This allows for a fully-verifying Wavelet node to run on hardware that simply has a minimum of 512MB of dynamic memory to spare, such as a smartphone.

Definition 5.4. Global-scale. Should liveness faults occur, the protocol naturally adjusts its own system parameters to take into account the load of the network to prevent further liveness faults from occurring. Benchmarks report a 5-second windowed moving average of 31,240 transactions being finalized per second on a network comprised of 240 nodes. Nodes utilize consumer-grade hardware (2vCPUs, 4GB RAM) deployed on a Kubernetes cluster on DigitalOcean with an average communication latency of 220 milliseconds alongside 2% packet loss simulated with the Linux tool **tc**(**1**). Each individual node was restricted to only hoist at most one single Wavelet instance, contained as a Kubernetes pod, and at most exert outbound traffic at a rate of 890kb/s. Each Wavelet instance contains a Snowball instance for finalizing rounds parameterized with $k = 10$, $\alpha = 0.8$, and $\beta = 150$ with full cryptographic signature verification performed for each individual incoming transaction. Each Wavelet instance is also parameterized with system parameters DEPTH_DIFF = 10, MAX_NUM_PARENTS = 32, MIN_DIFFICULTY = 8, and DIFFICULTY_SCALE = 0.5. The benchmark task was to have all nodes create and flood the network with batch transactions individually comprised 40 individual stake placement transactions to consider the latencies each individual node faces in maintaining their state S running Wavelet. Before statistics were collected, a warm-up period of 25 minutes was taken into consideration as benchmarks were being run. In a network where communication latency is negligible for the sole purposes of a hosting together a private blockchain, Wavelet is able to sustain finalizing on average over 180,000 transactions per second with the very same benchmark task.

Definition 5.5. Smart contracts. Given that the ordering and application of transactions is consistent across all nodes, Turing-complete smart contracts may safely be supported and deterministically executed. As a first step, WebAssembly smart contracts are supported with deterministic software-emulated floating-point and deterministic control-flow-graph-based gas counting. Before the invocation of a smart contract function, a snapshot of the memory of the interpreter is stored in-memory. After the invocation of a smart contract function, the difference between the post-invocation interpreter's memory and pre-invocation interpreter's memory snapshot is persisted onto the ledger.

Definition 5.6. Proof of stake. Recall the Snowball consensus protocol $\mathcal{P}_{\text{query}}$, and the proof-of-stake mechanism denoted in Section 3.3. Convolving R_k over S_k weighs progress over eventual termination of $\mathcal{P}_{\text{query}}$ through stakes of virtual currency, allowing for Wavelet variant W to naturally establish Sybil resilience.

Definition 5.7. Fair incentives. Recall the reward distribution scheme denoted in Section 3.4. The protocol naturally requires validators to have to put effort into assisting with the settlement of consensus rounds to prevent the *nothing-at-stake* problem, and defers the *rich gets richer, poor gets poorer* dilemma from occurring.

Definition 5.8. Decentralized governance. The protocol forms a network topology where wealth is very evenly distributed amongst nodes to encourage a decentralized form of governance. A guaranteed irreversibility of history allows for the establishment of protocol-wide events such as incremental protocol updates and system parameter tuning.

6 Conclusion

A decade has passed since the very first introduction of Bitcoin, with its genesis springing forward an ever-growing amount of new technologies and applications. The journey to amend and generalize the framework which Bitcoin sits upon has led the authors of this paper to introduce a new practical family of consensus protocols; to advance Satoshi Nakamoto's vision towards a decentralized future.

Wavelet emphasizes minimal configuration, minimal resource consumption, and minimal system requirements to prohibit only the rich from benefiting from such a decentralized network. An emphasis was also made to allow for a large diversity of decentralized applications to be built, adopted, and explored.

A small set of conjectures were also introduced to the reader throughout this paper, which upon exploration easily lends itself towards the development of promising, exciting, new mathematical toolings and frameworks.

One last problem towards the development of a general-purpose ledger framework has yet to be addressed however: it is trivial that over time, node states S may grow far too large, making disk space a bottleneck. The authors would like to conclude that a solution to such a problem has already been thought of, which will be introduced in a forthcoming paper titled **Perlin**.

Acknowledgements

The authors would like to acknowledge Satoshi Nakamoto for his work on Bitcoin, heavily impacting the authors lives from as early as the early 2000s. The authors would also like to acknowledge Dimitris Papadopolous, Foteini Baldimtsi, and Josh Lind for their valuable comments and feedback.

References

[1] Ingmar Baumgart and Sebastian Mies. S/kademlia: A practicable approach towards secure key-based routing. In *2007 International Conference on Parallel and Distributed Systems*, pages 1–8. IEEE, 2007.

[2] Ethan Buchman. *Tendermint: Byzantine fault tolerance in the age of blockchains*. PhD thesis, 2016.

[3] Kai Lai Chung and Paul Erdös. On the application of the borel-cantelli lemma. *Transactions of the American Mathematical Society*, 72(1):179–186, 1952.

[4] Pierre-Jacques Courtois, Frans Heymans, and David Lorge Parnas. Concurrent control with readers and writers. *Communications of the ACM*, 14(10):667–668, 1971.

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

[6] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 51–68. ACM, 2017.

[7] Leo A Goodman. Snowball sampling. *The annals of mathematical statistics*, pages 148–170, 1961.

[8] Richard Isaac. *The pleasures of probability*. Springer Science & Business Media, 2013.

[9] Dan Larimer. Delegated proof-of-stake consensus, 2018.

[10] Satoshi Nakamoto et al. Bitcoin: A peer-to-peer electronic cash system. 2008.

[11] Serguei Popov. On fast probabilistic consensus in the byzantine setting, 2019.

[12] Team Rocket. Snowflake to avalanche: A novel metastable consensus protocol family for cryptocurrencies, 2018.