Proof of history: what is it good for?

Victor Shoup*

May 7, 2022

The Solana whitepaper [Yak18b] describes, among other things, the notion of a *proof of history*, and claims that this notion may be used as a trusted source of time in order to improve the performance of proof-of-stake consensus protocols. The purpose of this note is to question this claim. More specifically, we shall argue that we see little evidence that using proof of history as a trusted source of time improves the performance of proof-of-stake consensus protocols in any substantive way. It is our hope that this note can serve as the starting point for an intellectually honest discussion on the potential merits of proof of history and to spur further research on the topic.

1 What is consensus?

Generally speaking, a *consensus protocol* enables a collection of networked nodes (computers) to agree on a collection of values (transactions), even if some nodes are faulty. In the blockchain space, the collection of values is typically a dynamically growing sequence of values (an append-only ledger). The essential security properties that any good consensus protocol should provide are *safety* and *liveness*:

- *Safety* means (roughly speaking) that all nodes agree on the same sequence of values (although at any instant in time, some nodes may only see a prefix of the sequence seen by other nodes).
- *Liveness* means (roughly speaking) that all honest (i.e., non-faulty) nodes make steady progress (to the extent that the network delivers messages sent between them).

Consensus protocols are designed to provide these properties under various assumptions, including *communication assumptions* (e.g, *synchronous*, *partially synchronous*, or *asynchronous* communication) *failure assumptions* (the number and type of faulty nodes, e.g., *crash* or *Byzantine*), and *set-up assumptions* (e.g., *permissioned*, *permissionless*, or *proof of stake*).

The consensus problem has a long history (see [PSL80, LSP82, Ben83, DLS88, BKR94, CL99, CKPS01, KS01, CKS05, RC05, MXC+16, DRZ18, PS18, BKM18, YMR+18, GAG+19, SDV19, GLT+20, CDH+21], to name just a few relevant papers).

*Email: victor@shoup.net. This note was written while the author was on leave from New York University at DFINITY. The views and opinions expressed herein are solely those of the author, and do not necessarily reflect the policies or positions of New York University or DFINITY.

Much of the early work on consensus worked exclusively in the *permissioned model*, where the *access structure* for the protocol (the nodes in the network, their public keys, and their voting power) is typically established by a centralized authority.

Bitcoin [Nak08] was the first truly *permissionless* consensus protocol, and is based on a proof of work. In a permissionless protocol, there really is no access structure — nodes are free to join the network and participate in the protocol without precondition or permission. At least in theory, such permissionless protocols attain the highest degree of decentralization possible. However, Bitcoin and other proof-of-work-based protocols suffer from certain disadvantages, such as their enormous energy waste and poor performance.

In recent years, a compromise approach, called *proof of stake*, has emerged (e.g., Ethereum "2.0" [EthPOS], Cardano [Card], Algorand [GHM+17], Solana [Yak18b], Internet Computer [DFI22]). In this approach, the access structure of the protocol is dynamically determined by a decentralized mechanism based on *staking* cryptocurrency in the network. A proof-of-stake consensus protocol allows the access structure of the protocol to change over time; however, at distinct intervals in time, it typically employs what is essentially a permissioned consensus protocol, with the access structure for the protocol at each interval determined by a decentralized staking mechanism. Conversely, a permissioned consensus protocol can typically be converted to one based on proof of stake, simply by using a decentralized staking mechanism to define the access structure for the protocol at different intervals in time.

There is some inconsistency in the literature with regard to the terms *permissioned*, *permissionless*, and *proof of stake*. While some authors treat *proof of stake* as a special case of *permissionless*, for our purposes here, we treat *permissionless* and *proof of stake* as very distinct types of set-up assumptions. In our opinion, *proof of stake* is more closely related to *permissioned* than it is to *permissionless*. Indeed, a proof-of-stake protocol enjoys many of the benefits, in terms of efficiency, of permissioned protocols; moreover, while it is less centralized than a traditional permissioned protocol, it is not nearly as decentralized as a truly permissionless protocol, such as Bitcoin.

Solana's consensus protocol is a proof-of-stake protocol, and at its core is based on a permissioned protocol; hence, the characteristics of this core protocol may be legitimately compared to other permissioned protocols in the literature.

2 What is proof of history?

The basic idea is this. Let H be a cryptographically strong hash function, such as SHA-256 [NIST15]. Starting from an initial value s_0, one party P, called the "prover", can compute a sequence of values $s_1, \ldots, s_N$ (a so-called "hash chain") as follows:

$$s_i \coloneqq H(s_{i-1}) \quad (i = 1, \ldots, N).$$

Let $1 \leq k < N$ be a parameter (discussed below), and assume that N is of the form $N = qk$. Suppose now that P sends the values

$$s_k, s_{2k}, \ldots, s_{qk} = s_N$$

to another party V, called the "verifier".

We assume that V already knows the value s_0. The verifier V can then verify the values $s_k, s_{2k}, \ldots, s_{qk}$ sent by P by running the following algorithm:

for $j \leftarrow 1, \ldots, q$ do
    $s \leftarrow s_{(j-1)k}$
    repeat k times: $s \leftarrow H(s)$
    if $s \neq s_{jk}$ return `reject`
return `accept`

This algorithm returns `accept` if and only if the values sent by P were computed correctly.

The essential properties of this simple protocol are the following:

(1) The verification algorithm is trivially parallelizable: each of the q iterations of the outer loop may be run concurrently with one another.

(2) Because of the cryptographic strength of the hash function, it is fairly safe to assume that the computation of the prover cannot be parallelized, and requires N sequential computations of the hash function H.

Because of the (apparently) inherently sequential nature of the prover's computation, a proof of history seems to provide a somewhat reliable proof of the passage of time. However, the amount of time that has passed implied by such a proof depends on the speed of the hardware available to the prover. We may therefore identify two weaknesses with using proof of history as a way of proving the passage of time:

(1) The amount of time that has "provably passed" seems very inexact — that is, a proof of history is inherently a very imprecise clock.

(2) The amount of computation required to verify such a proof of history is very high (even though this work can be parallelized).

The above mechanism is described in the Solana whitepaper [Yak18b]; however, the idea has been around a while — see, for example, Section 3.1 of [LW15], where the *identical* mechanism is described.

2.1 Augmented proof of history

As described in [Yak18b], the basic proof of history mechanism may be augmented so that it may also be used to commit to a sequence of transactions. Suppose that for some set $I \subseteq \{1, \ldots, N\}$, the prover has a collection $\{t_i\}_{i \in I}$ of transactions t_i. Typically, the number $|I|$ of such transactions will be much smaller than N. The prover may fold these transactions into its proof of history, so that for $i \in I$, it computes

$$s_i \coloneqq H(s_{i-1}, t_i).$$

The prover will also transmit the collection of transactions $\{t_i\}_{i \in I}$ as a part of its proof of history so that the verifier may also perform the corresponding hash computations. Such an augmented proof of history acts as a proof of the passage of time, as well as a commitment to the collection of transactions $\{t_i\}_{i \in I}$.

Note that the use of hash chains to commit to a sequence of transactions is a concept that is used by essentially all blockchains.

2.2 Proof of History vs Verifiable Delay Function

The basic notion of proof of history is related to, but not the same as, the notion of a *verifiable delay function (VDF)*. What are called VDFs in the literature (see [BBBF18, Wes18, Pie18, BBF18]) require that the verification algorithm takes much less computation than that of the prover, *without relying on parallelization*. Indeed, [BBBF18] characterizes mechanisms such as proof of history as "pseudo-VDFs".

The paper [BBBF18] describes a number of applications of VDFs in decentralized systems. One application of VDFs to proof-of-stake consensus is as a *trusted source of randomness*, which, among other things, can be used to randomly select a small validation committee from a larger set of nodes. Another application of VDFs to proof-of-stake consensus is as a "computational timestamp" to defend against "long range attacks". However, as pointed out in [BBBF18], this use of VDFs is a very fragile defense, because of the inherent imprecision of a VDF as a clock. VDFs have also been shown to be useful in the construction of truly permissionless consensus protocols — for example, the Chia blockchain [CP19] makes essential use of VDFs to build a permissionless consensus protocol similar to Bitcoin, with no staking, but based on *proof of space* rather than *proof of work*.

3 What is proof of history good for?

3.1 The Solana whitepaper

The Solana whitepaper [Yak18b] states that:

> *PoH* [proof of history] *is used to encode trustless passage of time into a ledger — an append only data structure. When used alongside a consensus algorithm such as Proof of Work (PoW) or Proof of Stake (PoS), PoH can reduce messaging overhead in a Byzantine Fault Tolerant replicated state machine, resulting inn* [sic] *sub-second finality times.*

Thus, it is claimed that proof of history can be used as a trusted source of time to improve the performance of proof-of-stake consensus. The whitepaper goes on to say that:

> *In depth description of the proposed Proof of Stake consensus algorithm is described in Section 5.*

Section 5 of [Yak18b] sketches a few elements of a protocol, which we can summarize as follows:

- At any point in time, one node, acting as a leader (a "generator") is producing a proof of history, augmented with a sequence transactions as described above in Section 2.1.
- This augmented proof of history is transmitted to validators, who verify the proof.
- At regular intervals, the leader and the validators each publish a signature on the proof of history up to that point — the leader's signature is used to authenticate the origin of the proof of history, and each validator's signature represents a vote on its validity.

  - Validators have stake and their votes are weighted according to their stake.
  - If a super-majority of validators (according to their stake-based weight) vote for a given proof within a certain amount of time, that proof and the corresponding sequence of transactions up to that point are considered finalized.

- If the validators detect that the leader is nonresponsive or misbehaving, an election is triggered and the validators vote for a new leader (again, with votes weighted by stake).

- Slashing (confiscation of stake) is used to punish validators who vote inconsistently.

The above summary is obviously lacking in many important details. However, in our opinion, the "in depth description" in Section 5 of [Yak18b] is hardly any more informative; in particular, it is lacking:

- a clear and concise formulation of a consensus protocol,
  - rather than just a very vague, high-level sketch of some ideas for a protocol;
- a precise statement of its properties,
  - such as, for example, standard security properties such as safety and liveness, as well as standard performance metrics such as communication complexity, latency, throughput, and computational complexity;
- an explicit statement of underlying assumptions,
  - such as, for example, standard communication and failure assumptions, and perhaps some assumption specifically related to proof of history, like an assumption on relative hashing power;
- a precise statement as to which assumptions imply which properties, let alone a proof of any such statement;
  - for example, even though it is stated that "this algorithm depends on messages eventually arriving to all participating nodes within a certain timeout", it is unclear as to whether such an assumption is needed for safety or liveness or both.

Because of this lack of detail, we are unable to properly assess Solana's consensus protocol. We shall therefore explore other online resources (below in Sections 3.2–3.6) to try to ascertain how using proof of history as a trusted source of time might be helpful in improving the performance of Solana's consensus protocol or of proof-of-stake consensus protocols more generally. However, we do note that at a few points in [Yak18b], it is briefly suggested (again, with no real details) that in addition to improving its performance, proof of history may be useful in defending against "long range attacks". As mentioned above in Section 2.2, VDFs (and pseudo-VDFs such as proof of history) are at best a fragile defense against such attacks. In any case, this is not germane to our question of how using proof of history as a trusted source of time helps in improving the *performance* of proof-of-stake consensus protocols.

3.2 The blog post "Proof of history: A clock for blockchain"

The blog post [Yak18a] attempts to give some intuition as to why using proof of history as a trusted source of time is helpful in improving the performance of proof-of-stake consensus protocols. It begins by stating that

> *One of the most difficult problems in distributed systems is agreement on time.*

While there may be some areas in distributed computing where this is true, there is no evidence, in our opinion, that "agreement on time" is a problem that has in any way hindered the development of efficient consensus protocols (at least those that are permissioned or based on proof of stake). We invite the reader to pick up almost any published paper on the consensus problem (see again [PSL80, LSP82, Ben83, DLS88, BKR94, CL99, CKPS01, KS01, CKS05, RC05, MXC+16, DRZ18, PS18, BKM18, YMR+18, GAG+19, SDV19, GLT+20, CDH+21]) to find any hint that "agreement on time" is a problem whose solution would in anyway lead to more efficient consensus protocols.

Note that while many practical consensus protocols that work in the partially synchronous model [DLS88] (such as the classical PBFT protocol [CL99], HotStuff [YMR+18], SBFT [GAG+19], ICC [CDH+21], and many others) do require a clock, they do not need highly-synchronized clocks: they only required that all correct nodes in the system have clocks that are running at roughly similar rates. The blog post [Yak18a] seems to suggest that something much more is required. In reference to the "agreement on time" problem, they say:

> *Decentralized networks have solved this problem with trusted, centralized timing solutions. For example, Google's Spanner uses synchronized atomic clocks between its data centers. Google's engineers synchronize these clocks to a very high precision and constantly maintain them.*

However, in our opinion, this is a red herring. Spanner [CDE+12] is a database that shards data across many sets of PAXOS [Lam98] state machines in data centers distributed around the world. (PAXOS is a consensus algorithm that assumes crash-failures only.) The reference to "synchronized atomic clocks" in [Yak18a] is a reference to Google's TrueTime technology. As described very clearly in [Bre17]:

> *Many assume that Spanner somehow gets around CAP via its use of TrueTime, which is a service that enables the use of globally synchronized clocks. Although remarkable, TrueTime does not significantly help achieve CA ...*
>
> *One subtle thing about Spanner is that it gets serializability from locks, but it gets external consistency (similar to linearizability) from TrueTime.*

Here, the 'C' in 'CA' stands for *consistency*, which [Bre17] says "we can think of as serializability for this discussion". In database terminology, a schedule of operations is *serializable* if it is equivalent to a *serial order*, which is a schedule of non-overlapping transactions (each transaction consisting of a sequence of operations). This notion is also called *serial consistency* in [Gif81], where the notion of *external consistency* is defined as an even stronger type of consistency. Roughly speaking, an externally consistent schedule is equivalent to a serial

order that is consistent with the order in which transactions can be *observed* to commit. As characterized in [Lis93]:

> *A violation of external consistency occurs when the ordering of operations inside a system does not agree with the order a user expects.*

In relation to Spanner (and the corresponding cloud service Cloud Spanner), an easy to digest explanation of serializability and external consistency can be found in [Cloud1]:

> *Cloud Spanner provides 'serializability', which means that all transactions appear as if they executed in a serial order, even if some of the reads, writes, and other operations of distinct transactions actually occurred in parallel. Cloud Spanner assigns commit timestamps that reflect the order of committed transactions to implement this property. In fact, Cloud Spanner offers a stronger guarantee than serializability called external consistency: transactions commit in an order that is reflected in their commit timestamps, and these commit timestamps reflect real time so you can compare them to your watch. Reads in a transaction see everything that has been committed before the transaction commits, and writes are seen by everything that starts after the transaction is committed.*

More details on Spanner, TrueTime, and external consistency can be found in [Cloud2].

Thus, we see that, in this context, the "agreement on time" problem and Google's use of "synchronized atomic clocks" is all about solving the problem of providing *external consistency in a distributed database*, which, in our opinion, is a problem that has little or nothing to do with that of achieving consensus on a blockchain: for consensus, we just need safety and liveness, while external consistency imposes extra constraints related to the timestamps associated with transactions and their relation to *time as measured by external observers*. Moreover, because of its inherent imprecision as a clock, it seems highly unlikely that proof of history could be used as a replacement for highly synchronized clocks to achieve external consistency in any application (blockchain-related or otherwise).

The blog post [Yak18a] gives as further "evidence" for the importance of the "agreement on time" problem a quote from Barbara Liskov [Lis93]:

> *Synchronized clocks are interesting because they can be used to improve the performance of distributed algorithms. They make it possible to replace communication with local computation.*

The blog post [Yak18a] then goes on to say that solving the "agreement on time" problem would unlock an "enormous wealth of distributed systems optimizations" leading to "a high throughput, high performance blockchain". Again, in our opinion, this is a red herring. While [Lis93] discusses the "agreement on time" problem with various applications, such as message delivery protocols and session key exchange protocols, there is nothing in [Lis93] to suggest that there is any relationship between the "agreement on time" problem and the consensus problem. The closest it comes to that is the section on Commit Windows, which discusses the use of synchronized clocks within replicated systems; however, just as with

the Spanner example above, synchronized clocks are only used in this application to achieve external consistency.

The blog post [Yak18a] also claims that proof of history is a verifiable delay function (VDF):

> *The Proof of History is a high frequency Verifiable Delay Function. A Verifiable Delay Function requires a specific number of sequential steps to evaluate, yet produces a unique output that can be efficiently and publicly verified.*

However, as we noted above in Section 2.2, proof of history is not really a VDF, in the sense that proofs cannot be efficiently verified: to verify a proof of history, the verifier must perform the same computations as the prover, the only difference being that the verifier's computations can be parallelized. If we have somehow overlooked the real benefit of proof of history in Solana's consensus protocol, this raises the question: *could Solana's consensus protocol be improved by using a true VDF in place of its notion of proof of history?*

In the prologue to the blog post [Yak18a], as well as several other related blog posts, it is claimed that

> *Solana is the most performant permissionless blockchain in the world. On current iterations of the Solana Testnet, a network of 200 physically distinct nodes supports a sustained throughput of more than 50,000 transactions per second when running with GPUs.*

This claim is not backed up by any comparisons to other protocols. For example, one might compare the above claimed throughput performance to that of the MirBFT protocol [SDV19], which obtains a throughput of over 60,000 transactions per second on a 100-node network, without relying on any special-purpose hardware (experiments were run on the IBM Cloud service). Moreover, by design and as demonstrated in [SDV19], MirBFT's throughput degrades very slowly as more nodes are added. Even though MirBFT is a permissioned protocol, it may also be deployed (as indicated in [SDV19]) as part of a proof-of-stake system, and so the comparison is a fair one.

3.3 The blog post "8 innovations that make Solana the first web-scale blockchain"

In the blog post [Yak19a], it is claimed that there is a so-called "clock problem" that has apparently bedeviled the blockchain industry and that proof of history is a solution to this problem. To quote from [Yak19a]:

> *Today's blockchain-based networks have a clock problem ...*
>
> *POH* [proof of history] *is a solution to the clock problem ...*
>
> *Whereas other blockchains require validators to talk to one another in order to agree that time has passed, each Solana validator maintains its own clock by encoding the passage of time in a simple SHA-256, sequential-hashing verifiable delay function (VDF).*

It is indeed true that some consensus protocols do communicate with one another in order agree that time has passed. For example, this is done in the classic PBFT protocol [CL99] as well as more recent consensus protocols such as HotStuff [YMR+18]. In such leader-based consensus protocols, this is typically done to detect when a leader is nonresponsive and to take appropriate action to elect a new leader. However, these protocols communicate with each other for a number of different reasons, and in terms of communication bandwidth and latency, the fraction of communication performed in service of "agreeing that time has passed" is typically a negligible part of the whole. We therefore conclude that, in terms of consensus protocols, there simply is no "clock problem" that needs to be solved — in our opinion, this is yet another red herring.

Not only does it seem unlikely that proof of history can improve the communication complexity of consensus protocols in general, we do not see any evidence that it improves the communication complexity of Solana's own consensus protocol in particular. Indeed, it is not clear to us that the communication complexity of Solana's consensus protocol is any better than that of any of the existing consensus protocols in the literature, simply because we are not aware of any concrete and verifiable claims made in any available documentation relating to the communication complexity of this protocol. Moreover, we speculate that to the extent that Solana's consensus protocol does exhibit good communication complexity, this is more likely a consequence of other techniques, such as their use of erasure-coded broadcast [Yak19d], which is a standard technique (see, for example, [CT05]). One reason erasure-coded broadcast can improve the practical performance of a leader-based consensus protocol is that it helps to alleviate the bandwidth bottleneck at the leader, who has to disseminate blocks to all nodes. This bottleneck-at-the-leader phenomenon was observed in [MXC+16] and [SDV19], and both papers address it in somewhat different ways, with [MXC+16] making use of erasure-coded broadcast (among other tools). Both [MXC+16] and [SDV19] back up all of their claims with both detailed theoretical analysis and extensive empirical results, neither of which appears to be available for Solana's protocol.

3.4 The blog post "A new architecture for a high performance blockchain"

In the blog post [Yak19b], a claim that is made that is certainly more germane to the relationship between clocks and consensus:

> *In general, there are two ways that classical distributed systems have dealt with clocks. Messages are time stamped by the sender and the timestamp is signed. Nodes drop messages that are either too old or too new. This calculation is based on the difference between the timestamp and the local clock. The second approach is that every state transition has a local timeout before it expires. On Tendermint, for example, a precommit state has a one second timeout. The next block producer can try to propose the next block, but all the nodes in the network will wait for at least 1 second from the start of the precommit state transition before considering the new proposal.*

And a bit later in [Yak19b]:

> *The Solana protocol has no dependencies on local computer clocks or local timeouts between state transitions beyond the Verifiable Delay Function. Instead, the VDF ensures that each block producer proves they have waited a sufficient amount of time, and the network moves forward. Unlike Tendermint, the next block producer has to locally generate a portion of the VDF until the scheduled slot.*
>
> *This means that receiving nodes can begin state transition as soon as the message is received, because they have a cryptographic proof that the producer obeyed the protocol delays.*

In our opinion, the criticism here of Tendermint [BKM18] is a strawman. The issue here is that of *optimistic responsiveness*, a notion introduced in [PS18], that, roughly speaking, says that the protocol will run *as fast as the network will allow*, at least under appropriate "optimistic" conditions (such as when the leader in a leader-based protocol is honest and the network is well behaved). While it is well known that Tendermint is not optimistically responsive, other protocols are, including classical PBFT [CL99], HotStuff [YMR+18], SBFT [GAG+19], and ICC [CDH+21]. These optimistically responsive protocols solve perfectly well the problem of avoiding unnecessary delays *without* the need for performing all of the intensive computations required for proof of history. Indeed, it is hard to see how any consensus protocol based on proof of history can possibly run as fast as the *network* will allow, since — by definition — the nodes in the protocol are busily wasting a significant amount of time computing hash chains.

3.5 The blog post "Tower BFT: Solana's high performance implementation of PBFT"

The blog post [Yak19c] states that:

> *Solana implements a derivation of PBFT, but with one fundamental difference. Proof of History (PoH) provides a global source of time before consensus. Our implementation of PBFT uses the PoH as the network clock of time, and the exponentially-increasing timeouts that replicas use in PBFT can be computed and enforced in the PoH itself.*

The exponentially-increasing timeouts in PBFT referred to here are the timeouts used to trigger a view change when a leader appears to be unresponsive. The purpose of increasing the timeouts in this way is essentially to ensure that the protocol makes progress, even if its initial estimate for maximum network delay is too small.

The blog post [Yak19c] claims that Solana's consensus protocol is a "derivation of PBFT", and sketches some basic ideas of Solana's consensus protocol and how it differs from PBFT. First, unlike PBFT, but similar to, for example, HotStuff [YMR+18], it rotates the leader frequently at regular intervals. Second, it uses a voting strategy to finalize transactions that is based on *exponentially increasing lockout periods*, during which time a node cannot vote on a different branch. These lockout periods are measured in terms of rounds, each of which is supposed to take a certain amount of time based on proof of history.

This idea of exponentially increasing lockout periods (which, by the way, is never mentioned in the Solana whitepaper [Yak18b]) is a potentially interesting idea, but its motivation and benefits remain unclear, and it raises a number of questions, such as: *under precisely what conditions is liveness guaranteed to hold?* Another question, more relevant to our discussion here, is: *why not simply use ordinary (unsynchronized) clocks to mark the passage of time?* This could presumably be done by standard techniques, whereby each node casts a vote to indicate that a certain amount of time, encompassing one "epoch", has passed (according to its local clock), and moves to the next "epoch" after seeing quorum of such votes (as just one recent example of this, see [CPS18]). Casting these votes would obviously cost a bit more in communication; however, it is not at all clear to us if the communication cost of these votes would be any more than the communication cost of all of the other voting going on in Solana's consensus protocol.

3.6 Other online documentation

Some aspects of Solana's consensus protocol are also hinted at in the online documentation at `https://docs.solana.com`, specifically:

- `https://docs.solana.com/cluster/overview`
- `https://docs.solana.com/cluster/synchronization`
- `https://docs.solana.com/cluster/leader-rotation`
- `https://docs.solana.com/cluster/fork-generation`
- `https://docs.solana.com/cluster/managing-forks`
- `https://docs.solana.com/cluster/turbine-block-propagation`
- `https://docs.solana.com/cluster/vote-signing`
- `https://docs.solana.com/cluster/stake-delegation-and-rewards`

This documentation (specifically, `https://docs.solana.com/cluster/managing-forks`) does say a few words about exponentially increasing lockout periods, which, as discussed above in Section 3.5, is not mentioned in the Solana whitepaper [Yak18b]. However, just like that whitepaper, this documentation is horribly lacking in details, suffering from the same lack of specificity that the whitepaper suffers from, as discussed above in Section 3.1 (namely, an underspecified protocol, no statement of its properties, no statement of underlying assumptions, no analysis or proofs).

4 Conclusion

We have observed that if one uses proof of history as a way of proving the passage of time, the amount of time that has "provably passed" seems very inexact, and the amount of computation required to verify such a proof of history is very high. We argued that we see little evidence that using proof of history as a trusted source of time is helpful in improving the performance of proof-of-stake consensus protocols.

- We examined the possibility that using proof of history as a trusted source of time could somehow dramatically reduce the communication complexity of proof-of-stake consensus protocols. We observed, however, that the communication complexity that other protocols devote to marking the passage of time is typically insignificant. Moreover, there is no clear evidence to suggest that proof of history plays any significant

role in improving the communication complexity of Solana's own consensus protocol, or that the communication complexity of this protocol is any better than that of any of the existing consensus protocols in the literature. Indeed, we speculate that to the extent that Solana's consensus protocol does exhibit good communication complexity, this is more likely a consequence of other techniques.

- We also examined the possibility that using proof of history as a trusted source of time could somehow avoid the delays introduced by consensus protocols, such as Tendermint, that do not enjoy the property of optimistic responsiveness (i.e., run as fast as the network will allow). We observed, however, that there are a number of other protocols that are optimistically responsive, including classic PBFT and several other protocols — these protocols avoid unnecessary delays without relying on proof of history. Moreover, it is hard to see how any consensus protocol based on proof of history can possibly run as fast as the *network* will allow, since — by definition — the nodes in the protocol are busily wasting a significant amount of time computing hash chains.

We observed that proof of history is not really a verifiable delay function (VDF), as proof-of-history verification is computationally expensive, unlike verification in a true VDF. If we have somehow overlooked the real benefit of proof of history in Solana's consensus protocol, a natural question to ask is whether their consensus protocol could be improved by using a true VDF in place of proof of history. We also observed that while VDFs do have legitimate uses in the design of proof-of-stake consensus protocols, those uses mainly derive from using VDFs as a trusted source of randomness (rather than time), and as a (fragile) defense against "long range attacks". VDFs also have been shown to be useful in the construction of truly permissionless consensus protocols — for example, the Chia blockchain [CP19] makes essential use of VDFs to build a permissionless consensus protocol similar to Bitcoin, with no staking, but based on *proof of space* rather than *proof of work*.

We observed that one interesting and perhaps novel idea in Solana's consensus protocol is that of using exponentially increasing lockout periods, during which time a node cannot vote on a different branch. This idea seems worthwhile to explore; however, its motivation, benefits, and security properties remain unclear; moreover, it is also unclear why using ordinary (unsynchronized) clocks together with standard techniques to agree on the passage of time would not be sufficient to implement this idea.

Final thoughts. We freely admit that our critique of proof of history and its use as a trusted source of time to improve the performance of proof-of-stake consensus is limited by a lack of information, even though we did our best to understand this issue by examining all of the available online articles and documentation for Solana (but not the source code). If the use of proof of history as a trusted source of time can indeed be used to significantly improve the performance of proof-of-stake consensus protocols, then the consensus research community would undoubtedly welcome a scientific paper that includes a clear and concise formulation of such a consensus protocol, a precise statement of its properties, an explicit statement of underlying assumptions, and a precise statement as to which assumptions imply which properties (and hopefully a proof of such a statement). Moreover, a review by the research community of such a paper would only serve to increase the broader blockchain community's confidence in the security of the protocol. Indeed, given the current lack of

published details and analysis, the lack of any meaningful peer review, and the inherent subtlety of designing secure consensus protocols, in our opinion, there currently seems to be little reason to have much confidence in the security of Solana's consensus protocol (see, for example, Section 2 of [CV17] for more on this general principle). It would also be immensely helpful to the research community to have a well-designed experimental analysis of the performance of such a protocol, relative to other proof-of-stake consensus protocols, in order to gain a better appreciation of any of its potential performance benefits.

Acknowledgments

We would like to thank Christian Cachin for feedback on an earlier draft of this document.

References

[BBBF18] D. Boneh, J. Bonneau, B. Bünz, and B. Fisch. Verifiable delay functions. Cryptology ePrint Archive, Report 2018/601, 2018. `https://ia.cr/2018/601`.

[BBF18] D. Boneh, B. Bünz, and B. Fisch. A survey of two verifiable delay functions. Cryptology ePrint Archive, Report 2018/712, 2018. `https://ia.cr/2018/712`.

[Ben83] M. Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols (extended abstract). In R. L. Probert, N. A. Lynch, and N. Santoro, editors, *Proceedings of the Second Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, Montreal, Quebec, Canada, August 17-19, 1983*, pages 27–30. ACM, 1983.

[BKM18] E. Buchman, J. Kwon, and Z. Milosevic. The latest gossip on BFT consensus, 2018. arXiv:1807.04938, `http://arxiv.org/abs/1807.04938`.

[BKR94] M. Ben-Or, B. Kelmer, and T. Rabin. Asynchronous secure computations with optimal resilience (extended abstract). In J. H. Anderson, D. Peleg, and E. Borowsky, editors, *Proceedings of the Thirteenth Annual ACM Symposium on Principles of Distributed Computing, Los Angeles, California, USA, August 14-17, 1994*, pages 183–192. ACM, 1994.

[Bre17] E. Brewer. Spanner, TrueTime & The CAP Theorem, 2017. `https://storage.googleapis.com/pub-tools-public-publication-data/pdf/45855.pdf`.

[Card] (Re)introduction into Cardano. `https://developers.cardano.org/docs/stake-pool-course/introduction-to-cardano/`.

[CDE+12] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, W. C. Hsieh, S. Kanthak, E. Kogan, H. Li, A. Lloyd, S. Melnik, D. Mwaura, D. Nagle, S. Quinlan, R. Rao, L. Rolig, Y. Saito, M. Szymaniak, C. Taylor, R. Wang, and D. Woodford. Spanner: Google's globally-distributed

database. In C. Thekkath and A. Vahdat, editors, *10th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2012, Hollywood, CA, USA, October 8-10, 2012*, pages 251–264. USENIX Association, 2012. URL https://www.usenix.org/conference/osdi12/technical-sessions/presentation/corbett. https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/39966.pdf.

[CDH+21] J. Camenisch, M. Drijvers, T. Hanke, Y.-A. Pignolet, V. Shoup, and D. Williams. Internet computer consensus. Cryptology ePrint Archive, Report 2021/632, 2021. https://ia.cr/2021/632.

[CKPS01] C. Cachin, K. Kursawe, F. Petzold, and V. Shoup. Secure and efficient asynchronous broadcast protocols. In J. Kilian, editor, *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, volume 2139 of *Lecture Notes in Computer Science*, pages 524–541. Springer, 2001.

[CKS05] C. Cachin, K. Kursawe, and V. Shoup. Random oracles in constantinople: Practical asynchronous byzantine agreement using cryptography. *J. Cryptol.*, 18(3):219–246, 2005.

[CL99] M. Castro and B. Liskov. Practical Byzantine fault tolerance. In M. I. Seltzer and P. J. Leach, editors, *Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI), New Orleans, Louisiana, USA, February 22-25, 1999*, pages 173–186. USENIX Association, 1999. URL https://dl.acm.org/citation.cfm?id=296824.

[Cloud1] Cloud Spanner (transactions). https://cloud.google.com/spanner/docs/transactions.

[Cloud2] Cloud Spanner documentation (TrueTime and external consistency). https://cloud.google.com/spanner/docs/true-time-external-consistency.

[CP19] B. Cohen and K. Pietrzak. The Chia network blockchain, 2019. https://www.chia.net/assets/ChiaGreenPaper.pdf.

[CPS18] T.-H. H. Chan, R. Pass, and E. Shi. Pala: A simple partially synchronous blockchain. Cryptology ePrint Archive, Report 2018/981, 2018. https://ia.cr/2018/981.

[CT05] C. Cachin and S. Tessaro. Asynchronous verifiable information dispersal. In P. Fraigniaud, editor, *Distributed Computing, 19th International Conference, DISC 2005, Cracow, Poland, September 26-29, 2005, Proceedings*, volume 3724 of *Lecture Notes in Computer Science*, pages 503–504. Springer, 2005.

[CV17] C. Cachin and M. Vukolic. Blockchain consensus protocols in the wild, 2017. arXiv:1707.01873, http://arxiv.org/abs/1707.01873.

[DFI22] The DFINITY Team. The Internet Computer for geeks. Cryptology ePrint Archive, Report 2022/087, 2022. https://ia.cr/2022/087.

[DLS88] C. Dwork, N. A. Lynch, and L. J. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, 1988.

[DRZ18] S. Duan, M. K. Reiter, and H. Zhang. BEAT: asynchronous BFT made practical. In D. Lie, M. Mannan, M. Backes, and X. Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 2028–2041. ACM, 2018.

[EthPOS] Ethereum: proof of stake, 2022. `https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/`.

[GAG+19] G. Golan-Gueta, I. Abraham, S. Grossman, D. Malkhi, B. Pinkas, M. K. Reiter, D. Seredinschi, O. Tamir, and A. Tomescu. SBFT: A scalable and decentralized trust infrastructure. In *49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2019, Portland, OR, USA, June 24-27, 2019*, pages 568–580. IEEE, 2019.

[GHM+17] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. Cryptology ePrint Archive, Report 2017/454, 2017. `https://eprint.iacr.org/2017/454`.

[Gif81] D. Gifford. *Information Storage in a Decentralized Computer System*. PhD thesis, Stanford University, 1981. `http://www.bitsavers.org/pdf/xerox/parc/techReports/CSL-81-8_Information_Storage_in_a_Decentralized_Computer_System.pdf`.

[GLT+20] B. Guo, Z. Lu, Q. Tang, J. Xu, and Z. Zhang. Dumbo: Faster asynchronous BFT protocols. In J. Ligatti, X. Ou, J. Katz, and G. Vigna, editors, *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, pages 803–818. ACM, 2020.

[KS01] K. Kursawe and V. Shoup. Optimistic asynchronous atomic broadcast. Cryptology ePrint Archive, Report 2001/022, 2001. `https://ia.cr/2001/022`.

[Lam98] L. Lamport. Proving possibility properties. *Theor. Comput. Sci.*, 206(1-2):341–352, 1998.

[Lis93] B. Liskov. Practical uses of synchronized clocks in distributed systems. *Distributed Comput.*, 6(4):211–219, 1993. `https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.1089.5025&rep=rep1&type=pdf`.

[LSP82] L. Lamport, R. E. Shostak, and M. C. Pease. The Byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

[LW15] A. K. Lenstra and B. Wesolowski. A random zoo: sloth, unicorn, and trx. Cryptology ePrint Archive, Report 2015/366, 2015. `https://ia.cr/2015/366`.

[MXC+16] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song. The honey badger of BFT protocols. In E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 31–42. ACM, 2016.

[Nak08] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. `https://bitcoin.org/bitcoin.pdf`.

[NIST15] National Institute of Standards and Technology. Secure hash standard (SHS). Federal Information Processing Publication 180-4, 2013. `https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf`.

[Pie18] K. Pietrzak. Simple verifiable delay functions. Cryptology ePrint Archive, Report 2018/627, 2018. `https://ia.cr/2018/627`.

[PS18] R. Pass and E. Shi. Thunderella: Blockchains with optimistic instant confirmation. In J. B. Nielsen and V. Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 3–33. Springer, 2018.

[PSL80] M. C. Pease, R. E. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, 1980.

[RC05] H. V. Ramasamy and C. Cachin. Parsimonious asynchronous byzantine-fault-tolerant atomic broadcast. In J. H. Anderson, G. Prencipe, and R. Wattenhofer, editors, *Principles of Distributed Systems, 9th International Conference, OPODIS 2005, Pisa, Italy, December 12-14, 2005, Revised Selected Papers*, volume 3974 of *Lecture Notes in Computer Science*, pages 88–102. Springer, 2005.

[SDV19] C. Stathakopoulou, T. David, and M. Vukolic. Mir-bft: High-throughput BFT for blockchains, 2019. arXiv:1906.05552, `http://arxiv.org/abs/1906.05552`.

[Wes18] B. Wesolowski. Efficient verifiable delay functions. Cryptology ePrint Archive, Report 2018/623, 2018. `https://ia.cr/2018/623`.

[Yak18a] A. Yakovenko. Proof of history: A clock for blockchain, 2018. `https://medium.com/solana-labs/proof-of-history-a-clock-for-blockchain-cf47a61a9274`.

[Yak18b] A. Yakovenko. Solana: A new architecture for a high performance blockchain v0.8.13, 2018. `https://solana.com/solana-whitepaper.pdf`.

[Yak19a] A. Yakovenko. 8 innovations that make Solana the first web-scale blockchain, 2019. `https://medium.com/solana-labs/7-innovations-that-make-solana-the-first-web-scale-blockchain-ddc50b1defda`.

[Yak19b] A. Yakovenko. How Solana's proof of history is a huge advancement for block time, 2019. https://medium.com/solana-labs/how-solanas-proof-of-history-is-a-huge-advancement-for-block-time-178899c89723.

[Yak19c] A. Yakovenko. Tower BFT: Solana's high performance implementation of PBFT, 2019. https://medium.com/solana-labs/tower-bft-solanas-high-performance-implementation-of-pbft-464725911e79.

[Yak19d] A. Yakovenko. Turbine — Solana's block propagation protocol solves the scalability trilemma, 2019. https://medium.com/solana-labs/turbine-solanas-block-propagation-protocol-solves-the-scalability-trilemma-2ddba46a51db.

[YMR+18] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham. HotStuff: BFT consensus in the lens of blockchain, 2018. arXiv:1803.05069, http://arxiv.org/abs/1803.05069.