Solana Labs

Architectural Security Review and Report

November 4, 2019

Version: 1.0

Classification: For Public Release

Presented by:

Kudelski Security

Providentia Worldwide

Research Team

KUDELSKI SECURITY

Table of Contents

KUDELSKI SECURITY

KUDELSKI SECURITY

Tables and Figures

1 Revision History

Date	Version	Revision
2019-10-25	0.1	Initial. SRQ. AK.
2019-11-01	0.2	Findings update after initial reviews.
2019-11-04	0.3	Add Innovation Dimension Rankings and relocate Blockchain Trilemma to Appendix B.
2019-11-05	1.0	Final version. Attached Solana Whitepaper (2017) to PDF version for printing.
2019-11-12	1.0	Issued version classified for public release.

2 Executive Summary

Kudelski Security (“Kudelski”), the cybersecurity division of the Kudelski Group, was engaged by Solana Labs. (“Company”) client to conduct an external security assessment in the form of an architectural security evaluation of the core Solana innovations and platform.

The assessment was conducted onsite by the Kudelski Security Team and our trusted partners Providentia Worldwide at the Solana offices in Boulder, Colorado and San Francisco, California. Our analysis took place from September 30, 2019 to October 15, 2019 and focused on the following objectives:

- To help Solana better understand their security position with regard to cryptography and its usage in their platform;
- The security and risk position of the core innovations claimed by the Solana Labs team;
- The risk and design security posture of the Solana whitepaper as it relates to their current implemented platform and near-future deployments.

The work consisted of long-form whiteboard sessions with the core Solana development and business teams, detailed analysis of their published documentation, and assessments as to the degree of accuracy within their disclosures to date. The evaluation team did not conduct direct testing of the Solana platform as a part of this evaluation, but did go into great detail on the mechanisms of deployments undertaken to date as well as the specifics of those deployments, ability to scale and impact to security considerations.

2.1 Analyzed Deployment

One of the key aspects of this evaluation effort, which is different from many like it, is that the system exists in a state of flux between theoretical and architectural design, engineering implementation, functional testing, and scale testing. This means that for each component, the actual tested component may not align with all the others with regards to tested functionality, tested scale, design verification, and the like. This means that it is difficult to ascertain precisely how to measure the overall platform.

Nevertheless, we have endeavored to approach the system as a coherent whole. To do so, we often have to rely on currently deployed components which may not be implemented to production, full-performance-scale versions. We rely on tested analysis where we can, but since this audit is meant to focus on the risk and potential behaviors when faced with full production deployments either in permissioned (governed, centralized) or permission-less (decentralized, ungoverned) modes. Since we have only a small-scale, permissioned system to refer to at the time of writing, some of the analysis reflects this and we call the situation out in the details.

This is not meant to say that the platform cannot perform at higher levels, or that we impugn the results because of the deployed scale at the time. Rather we point out that where we have concerns with scale, we ask that we see additional testing and functionality at these levels to further verify our assumptions. Where the investigating team has not directly witnessed such tests in the past, we suggest that the Solana team itself is an excellent resource to explain their results and test designs in these cases.

Most of the remediation suggestions require additional testing, documentation, and case-evaluation. We recommend that the Solana team embark on these tests accordingly.

2.2 Decentralization and the “Lowest Common Denominator” Problem

For public, permission-less blockchains, there is often a resistance to exotic, expensive hardware as a base requirement for network participation. This means that there is a deliberate effort by the supporting community and thereby a technology bias such that smaller, less-performant computers are allowed and often set as a baseline by which the system is measured. For example, the length of time it should take to load a “full node” might be measured using only a 1Gbps network connection on a system with a 1.5Ghz clock speed and 2GB of memory. This would be set as a minimum bar for network participation and so becomes the “lowest common denominator” in all technical designs and implementations.

Solana adopts this same outlook with regards to their blockchain ecosystem.

We call this out in this section because there are times when the lowest common denominator results in lower performance or behavior by nodes and participants when there is no technical reason for there to be lower performance or behavior. This can impact security, scalability, availability, or other dimensions of the platform analysis.

For example, in a homogenous system with all nodes being equal and clock speeds of 2.4Ghz the Solana platform will outperform "the lowest common denominator" system only based on hardware with no changes to software or architecture. In a heterogenous deployment, like a permission-less system, the speed of the overall deployment will be limited to the slowest performing staked leader. This could be well below the capabilities of other participants, making their hardware and platform investments irrelevant and making stake-weight the only relevant leadership factor. By this design, a well-staked Raspberry Pi makes more sense than a GPU-accelerated dual-socket system.

2.3 Engagement Limitations

The timeframe for the engagement was limited to onsite discussions and research performed by the core team once the engagement was underway. The team ensured that they were well-versed in the current documentation for the platform, the innovation claims, and the whitepaper theory before the onsite engagement to better maximize the use of time. The desired time for delivery was within one calendar month of the start of the onsite engagement with regular updates in the interim.

The engagement was not able to conduct testing on live site components, on either production or development deployments of the platform. Likewise, several scenarios under investigation were not tested by the Solana team in their deployed embodiments and so limit the accuracy and commentary of this report to some degree. Where those situations occur, they are called out in the text and discussion in the following sections.

The report includes items called out in the initial statement of work signed with Kudelski in August 2019 that directly covers security and architectural issues, but does not focus on comparative analysis or additional evaluations or analyses.

2.4 Document Structure

As this is not a standard security evaluation with direct tests and results, but a document which analyzes claims and postures both in theoretical computer science as well as in real-world deployments, the structure of this document maps to the desires of the Client. Solana specifically requested that we report on their claims according to the schedule which they claim them and map them to a features and capabilities matrix popularized in the blockchain industry. These are illustrated in the section below.

3 General Platform Notes and Observations

Generally speaking, the Solana platform includes several novel claims for blockchain technologies which serve to enhance the overall blockchain and digital ledger technology ecosystem from a security and managed-risk perspective. These include a demonstrable Proof of History consensus mechanism, provably ordered events, and block distribution capabilities. Each is explained in their sections below.

Like many blockchain organizations, it can sometimes be difficult to isolate the behavior of the platform and the chain from the ideologies underlying its construction in the first place. Concepts such as permissioned and permission-less chains, byzantine fault tolerance versus traditional resiliency models, staked election behaviors and the like are often concepts which mire in discussions of risk and security but tend to have more to do with the overall environment and ecosystems of their deployment than they do with the actual implementing technology. This is the case with Solana as well. The report will describe when these issues become murky in their sections below.

3.1 Technical Conventions for Discussion

Due to the nature of the technology, the terms for “clock” and “time” can be overloaded. For the purposes of this document:

- *“Clock Frequency*” refers to the cycles per second for a CPU or GPU in a modern computer.
- “C*lock”* alone refers only to the passage of event time styled as Proof of History and making up the core of the Solana innovative platform.
- *“Ticks”* refer to individual signed hash events which make up the frequency wave of the Proof of History *clock*
- “*Vector clocks*” refer to intermingled Proof of History clocks which are referenced in the whitepaper but have not been implemented.
- “*Time*” refers to wall clock time as experienced by human beings and not to any of the inventions, innovations, and whitepaper details in this document or in Solana discussions.

These conventions were generally agreed to by the team during our conversations and we find it useful to stick to them herein to make the technical assessments easier to comprehend and prevent misunderstandings in the findings.

3.2 Claims and Innovations

Solana has authored a computer science whitepaper which elucidates the theory and background of the platform. The Solana Whitepaper, written in 2017, serves as a foundational guide and architecture roadmap for the platform. The original is attached to this document as (Appendix C). While we do not comment on the theoretical claims beyond the scope of the security audit, we suggest reading the paper in its entirety to understand the overall objectives of the Solana platform and technology to help understand their risk and security posture in detail.

Additionally, as the Solana team has developed their platform and deployed their core technologies, they have created several components which they contend represent true innovation to the blockchain and digital ledger technology ecosystems generally. These are

documented on the Solana website, but we will include some limited discussion of each here for clarity purposes in reading the document. We reserve the right to quote liberally from their claims and will represent those *claims in italics* where necessary.

The core Solana claims are (numbering for reference in this document, not for priority or ranking of innovation importance or relevance):

- *Proof of History (POH)— a clock before consensus*
- *Tower BFT — a PoH-optimized version of PBFT*
- *Turbine — a block propagation protocol*
- *Gulf Stream — Mempool-less transaction forwarding protocol*
- *Sealevel — Parallel smart contracts run-time*
- *Pipelining — a Transaction Processing Unit for validation optimization*
- *Cloudbreak — Horizontally-Scaled Accounts Database*
- *Archivers — Distributed ledger store*

Each of the innovations is discussed in their own section in the details below.

4 Overall Security Posture

This section will outline the platform findings at a high level, with a focus towards the overall direction for the infrastructure as Solana moves forward. Detailed information between interacting components will be covered in this section, but specifics on component behavior will be outlined in the relevant sections below.

4.1 Innovation Quadrant

Made famous by Venture Scanner (https://www.venturescanner.com/overview/), the Solana team has asked for the report to speak directly to their positioning from a security and risk perspective on the Innovation Quadrant. Figure 1 shows the June 2019 version of the quadrant infographic as discussed by Venture Scanner with regards to general technology claims[1].

[1] For a lengthy discussion of the methodology behind the quadrants, see https://www.venturescanner.com/blockchain-technology/. We do not comment on the Venture Scanner accuracy here, but were asked to try and leverage visual mechanisms by which the core team could explain to investors and technologists expressing interest in the Solana platform to place their innovations along a comparative scale. Since this is not properly the responsibility of the security-only audit, we have left the discussion of the original rationale as an exercise for the reader.

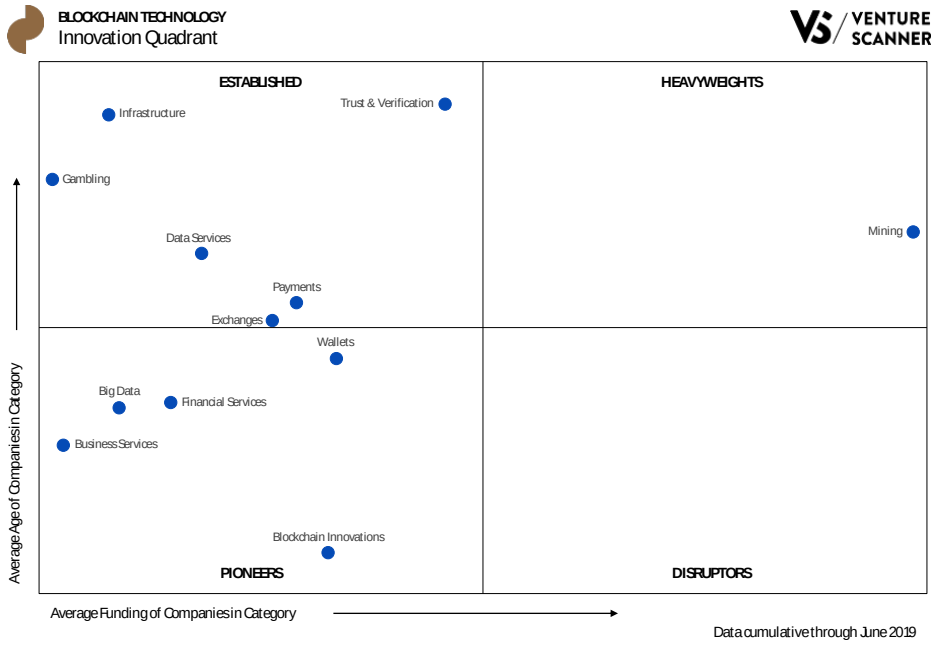


Figure 1: Blockchain Technology Innovation Quadrant (source Venture Scanner)

4.1.1 Overall Blockchain Innovation Quadrants

This section will put the positioning for each of the quadrants on a single graph with a short description of placement.

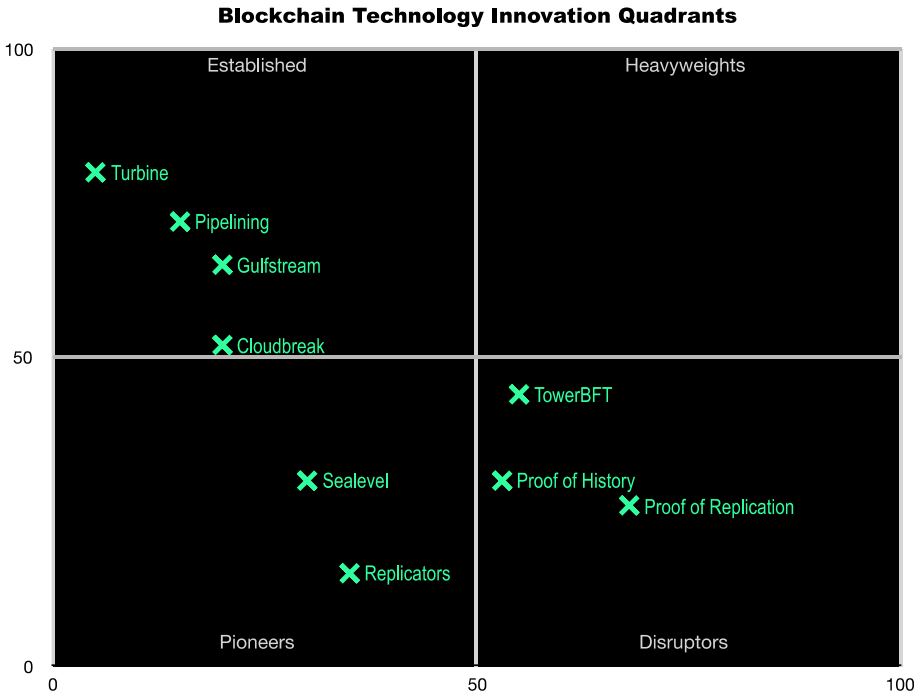


Figure 2: Solana Innovation Quadrants

The rankings here reflect a fairly subjective placement of the innovations in relation to one another and to other technologies which are employed in blockchain ecosystems. This is a difficult task since there are a variety of technologies for different purposes and several which have no direct antecedents in blockchain, but which are well-known in other disciplines.

We consider the Turbine, Pipelining, Gulfstream, and Cloudbreak technologies to be established concepts with a long history in similar deployments. These are not specific to blockchain however, and so do not edge towards disruption in the space.

Sealevel and Archivers provide unique capabilities to blockchain but are unproven in the long-term and have not seen enough production usage at scale and in common failure scenarios in long-lived deployments to understand their impact to the overall technology space yet. We see them as pioneering concepts which could disrupt and provide leadership computing in the blockchain ecosystem with time.

TowerBFT and its associated proofs of History and Replication provide a fully implemented and tested alternative to the prevailing consensus algorithms and verification protocols widely adopted in the blockchain ecosystems. While it is early yet to see how much of a leadership computing role they will play moving forward, it is clear that they represent disruptive implementations in the space.

4.2 Innovation Dimension Rankings

Much like Vitalik Buterin's famous Blockchain Trilemma[2], the Solana team wanted to see their innovations shown in light of their intended technical goals and against the security analysis of this investigation. Since the technology innovations do not lend themselves readily to direct comparison against one another since they aim to improve or change behavior in a variety of established areas they do not rank easily in the same plane. Likewise, since Solana recognizes that many improvements to a blockchain system require innovation in areas which might not directly impact the chain itself, but rather enable functionality or performance for components of the chain, ranking innovations directly on the Blockchain Trilemma is also difficult.

Rather, we have included rankings individually using the key components of the Blockchain Trilemma. We are calling this categorization the "Innovation Dimension Rankings".

Innovation Dimension Ranking Guidelines:

- All rankings are on a 100-point (percent %) scale.
- *Scalability* is bounded within the technology area for the innovation itself. This is to say that since scalability can be measured in a variety of ways depending on the perspective of the observer, we have deliberately constrained these rankings to measure within the specified goals for the innovation. For example, if there is an

[2] See Appendix B (Section 8) for a discussion and ranking of the Solana Innovations along Buterin's popular scale.

improvement to an in-memory data structure, then we are measuring the effectiveness of memory usage in a node, and not necessarily the capability for that data structure to scale horizontally across multiple platforms.

- *Security* aims to understand the security posture of the innovation itself and possibly against similar or competing technologies in that same area. For example, it would be reasonable to compare an in-memory data structure against other well-known in-memory data structures, but not against a distributed array database.
- *Decentralization* is a stated goal for all of Solana's innovations in that the platform aims to provide a high-performance, highly available, and permission-less platform for currency and tokens. Where there is always the potential to use the Solana platform in governed, centralized ways, the team wants to provide a competing technology to the well-known public blockchains like Bitcoin and Ethereum (among others). Thus, scoring well on the decentralization scale means that an innovation directly influences the ability of the technology towards permission-less decentralization of components.

The overall rankings are provided in the table below, with explanations located in the sections for the innovations.

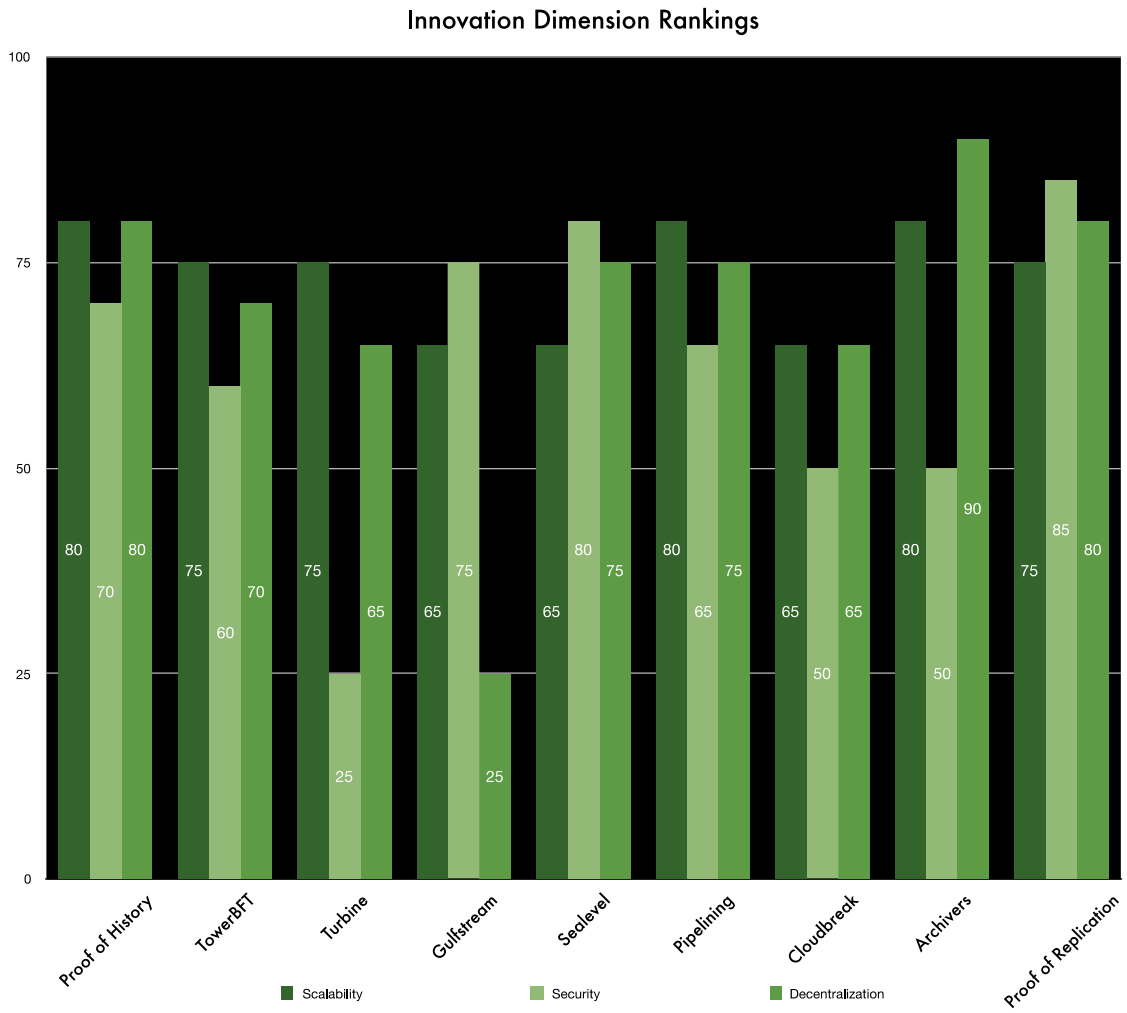


Figure 3: Innovation Dimension Rankings

4.3 Security Posture

Due to the nature of the investigation, and the division of the Solana platform into multiple “innovations”, it is difficult to characterize a single overarching posture. We can attest that the Solana team is security conscious at all levels and work hard to ensure that the platform is both flexible and secure. Generally speaking, the platform is primarily a design at this stage and while there is a functioning deployment, it is not deployed to scale and does not sustain the transaction volume which their testing claims. Likewise, where the design aims for a permission-less deployment in the long-term, all of the current nodes both in functional and test deployments have been controlled by the Solana team and so the specifics of behavior in heterogenous and permission-less implementations is conjecture. As such no full-system security posture can be reached due to no at-scale deployment available to be tested end-to-end. We are thus able to only comment on each individual component related to their individual security capabilities.

That said, we have taken the approach to categorize findings by the general case where that makes sense and is explained below. We localize findings to the innovation where the sphere of influence for that finding is best understood.

4.3.1 Key Takeaways

Solana aims to provide a blockchain for general token and currency support. Likewise, they designed the platform towards a fully decentralized and public deployment. This means that there are key components which must be present for trust of the platform and the assets it moves.

4.3.1.1 Theft

Prevention of theft and verifiably audit trust for the ledger are absolute must-haves from a security posture perspective. We did not find any indications that asset, token, or currency theft could occur through implementation of the chain and innovations themselves as outlined. We have not had an opportunity to audit the code itself as mentioned above, but the design is sound, solid, and trustable from both a security of asset and security of accounting ledger perspective.

4.3.1.2 Asset Vulnerability

We did not see any indication that assets in the form of tokens, incentive payments, currencies, smart contracts, or data in blocks were vulnerable to design inconsistencies or vulnerabilities. Likewise, we did not observe any functional behavior which would compromise the integrity of such assets.

Solana does not address the representation or mapping of digital assets to physical assets. While the platform does not prevent usage of tokens in this manner, it does not provide any functionality either to enhance or degrade behavior in these areas. As such we did not investigate or evaluate this aspect of asset management on the ledger.

4.4 Personnel

While the general purpose of this analysis is to focus on the technology as designed and implemented, it bears a short acknowledgement to the skills and knowledge of the Solana team. Solana has implemented a strategy for personnel which leverages individual talents to make leaps forward in technology, but with a constant attention towards backfill and knowledge transfer. In our discussions, the team functioned as a true team, with each member able to comment and enrich the discussion regardless of the particular topic. This means that the key human resource structure of the Solana team is well-positioned for growth and can tolerate the loss or departure of individuals without losing focus or capabilities in continued development and implementation of the platform.

This is generally a risk for small companies and startups, and it is remarkable that this situation is not the case for Solana. They have done an excellent job at conveying and distributing knowledge and training across their entire team.

4.5 All Findings

The table below illustrates all of the findings in this document by section and a quick assessment of claim validity. Detailed discussions of each finding, including exploits, likelihood, associated risks and mitigations are in the section referenced. In the table below, we list all findings in this document, with cross-references to the section for discussion. We include here as well the notion of Claim Validity for quick lookup.

4.5.1 Claim Validity

This is a quick analysis of whether the investigating team agrees with the claims as initially made by the Solana team, with the mitigation of an issue, or if that issue poses a large threat to the design and/or platform. Admittedly, this is a subjective analysis of many components, but we want to provide a quick location for looking at the overall posture by component. Since we are not evaluating a full production deployment, some of the claims present higher or lower risks as a result. Nevertheless, we find that putting supporting evidence in a single location makes it easier for using this document as a reference. Please refer to the more detailed discussion in each relevant section for more information.

Table 1: All Findings and Claims

Section	Finding	Scope	Risk	Remediable	Claim Validity
5.1	Fixed Transaction Sizes	General	Understood and acknowledged for all innovations	Possible	All claims based on fixed transaction size. Variable sizes should be investigated and understood.
5.2	Fixed Computational Frequencies	General	Understood	Possible	Needs additional investigation and verification.
5.3	General Resource Limitations	General	Understood	Possible	Needs additional investigation by component to better understand behaviors.
5.4	Unverified Transaction Handling	General	Understood and acknowledged for all innovations	Possible	Needs to be explicit that client code must handle these situations.
5.5	Verifiability of Functionality	General	Understood	Possible	Needs additional investigation by component to better understand behaviors.

Section	Finding	Scope	Risk	Remediable	Claim Validity
5.5.1	Planned Functionality	General	Variable	Possible	The platform design is a work in progress, and so the degree of risk and the validity of related claims varies.
5.5.2	Tested Functionality	General	Variable	Possible	While all components are tested and have a good rigor in test design, the actual functionality tested tends to vary by component and mature over time. Additionally, most testing is done under the auspices of devops code coverage[3] and planning, but more attention is needed to deployed testing especially as there is a pivot to permission-less.
5.6.1	SHA-256 Hash Chains	General	Understood and acknowledged for all innovations	Yes	Valid claims, modular approach allows changing to alternate methods easily.
5.6.2	CBC Chaining	General	Understood and acknowledged for all innovations	Yes	Valid claims, modular approach allows changing to alternate methods easily.
5.6.3	Quantum-Proof Cryptography	General	Understood and acknowledged for all innovations	Yes	Valid claims, modular approach allows changing to alternate methods easily.
5.7	Network Protocol Verification and Amplification	General	Understood and acknowledged for all innovations	Yes	Remediations will depend on the protocol.
6.2.2.1	Proof of History: Clock Frequency	Proof of History	Needs verification	Possible	Needs additional study.

[3] The Solana team makes a solid effort to bring code coverage and functional testing into their design principles. See https://github.com/solana-labs/solana#code-coverage and https://codecov.io/gh/solana-labs/solana for more details. We want to make sure that while we acknowledge and applaud the importance placed on design testing and code verification, there is no replacement for real-world battle scars and encourage extensive deployed, red-zone testing for all components.

Section	Finding	Scope	Risk	Remediable	Claim Validity
6.2.2.2	Proof of History: Vector Clocks		N/A	N/A	Claim removed for this effort.
6.3.2.1	Voting Assumptions	TowerBFT	Understood	Possible	Needs additional study.
6.3.2.2	Network Partitioning		Needs verification	Possible	Needs additional verification and validation to identify matrix of specific situations.
6.3.2.3	SHA-256 Hash Chains		Understood and acknowledged	Yes	Valid claims, modular approach allows changing to alternate methods easily.
6.4.2.1	Network Distribution	Turbine	Understood	Possible	Needs additional verification and validation to identify matrix of specific situations.
6.4.2.1.1	Bandwidth Overruns		Understood	Possible	Needs additional verification and validation to identify matrix of specific situations.
Figure 4	“Bump in the Wire”		Understood	Possible	Mitigation may vary by deployment.
6.4.2.1.2	Unidirectional Traffic Isolation		Understood	Possible	Mitigation may vary by deployment.
6.4.2.1.3	Network Partition: Failure Partitions		Needs verification	Possible	Mitigation may vary by deployment.
6.4.2.1.4	Network Partition: Administrative Partitions		Needs verification	Possible	Mitigation may vary by deployment.
6.5.2.1	“Fast Forward”	Gulfstream	Needs verification	Possible	Needs additional verification and validation to identify matrix of specific situations.
6.5.2.2	“Slow to a Crawl”		Needs verification	Possible	Needs additional verification and validation to identify matrix of specific situations.
6.5.2.3	“Unverified Masses”		Needs verification	Possible	Needs additional verification and validation to identify matrix of specific situations.

Section	Finding	Scope	Risk	Remediable	Claim Validity
6.6.2.1	Unverified Rust Implementation of eBPF	Sealevel	Needs external verification	Possible	Needs external validation and acceptance.
6.6.2.2	Account Management		Needs verification	Possible	Needs additional verification and validation to identify matrix of specific situations.
6.6.2.3	Sealevel: Auditability		Understood	Possible	Needs additional verification and validation to identify matrix of specific situations.
6.6.2.4	De-Optimizers		Needs verification	Possible	Needs additional verification and validation to identify matrix of specific situations.
6.6.2.5	All Accounts in Transaction		Needs verification	Possible	Needs additional verification and validation to identify matrix of specific situations.
6.7.2.1	Shared Components	Pipelining	Understood	Possible	Needs additional study. Some reliance on external forces may make some claims unprovable (compiler, OS, etc).
6.7.2.2	Parallel Pipelines		Understood	Possible	Needs additional study. Some reliance on external forces may make some claims unprovable (compiler, OS, etc).
6.8.2.1	Single Account Transactional Overload (“The iTunes Problem”	Cloudbreak	Needs verification	Possible	Needs additional study and verification.
6.8.2.2	Replication Reconstruction		Needs verification	Possible	Needs real-world demonstration at scale and under production behaviors – especially for production recovery and failure scenarios.

Section	Finding	Scope	Risk	Remediable	Claim Validity
6.8.2.3	Intentional Garbage Creation (“Million Voices Problem”		Needs verification	Possible	Needs a mitigation strategy and demonstration.
6.9.2.1	General Dependencies	Archivers	Understood	Possible	Archivers are still under active investigation by the team and so the claims here are not as strong.
6.9.2.2	“Replacing the Past” (“Doctor Who Attack”)		Understood	No	There is no enforceable mitigation for this, though the damage from the risk may not be great depending on the implementation. Requires external, verifiable testing to prove compliance to remediate.
6.9.2.3	CBC Chaining		Understood and acknowledged	Yes	Valid claims, modular approach allows changing to alternate methods easily.

5 General Findings and Claims

5.1 Fixed Transaction Sizes

While not a criticism specifically related to security safety, the assumptions and published data throughout the claims and whitepaper assume a fixed transaction size for all calculations. Since the system is relying on chained hashes to create both clock ticks and to verify those ticks as valid, moving the associated data involved in the hashing process is required[4]. For an implementation with varying data sizes by transaction the following scenarios may occur:

- Latency-based availability, denial-of-service, and resiliency compromises. This is especially true for network-centric Solana innovations such as Turbine and those applications which rely on them.

[4] See section 4.2 and following of the Solana whitepaper for a discussion of embedding and appending data to the transaction event stream.

- Large-buffer compromises. Specifically, in cases where transactions can be made arbitrarily large, it may be possible to consume bandwidth, increase latencies, and cause issues with internal memory management and transaction processors.
- Brand and claim compromises. Without specifically calling out the transaction size as measured and claimed in public sources, other users and testers of the network could experience varied behavior. This could substantially endanger the brand and claims of the Solana platform generally.
- Input-Output (IO) calculations are underestimated in many cases. Especially where more than one of the above scenarios may come into play (e.g. variously large transaction sizes in a geo-distributed network of validators), these potential findings may combine and compound one another[5].
- For the system to operate as designed, an enterprise must ensure that nodes which participate here are controlled to not introduce packet size discrepancies. In an open system this is not possible due to code changes, but in an authenticated node environment, this is.
- Permission-less testing has not been completed. While there is a testnet which is open to the public, the ends and stresses of these conditions have yet to be proven.

5.2 Fixed Computational Frequencies

Similar to the Transaction Size finding above, many of the calculations in the innovations and the Solana whitepaper make assumptions of homogeneity between clock frequency[6] on the systems. That is to say, CPUs of the same generation and frequency range might be said to work in generally the same ways and generally the same speeds, but the actual deployments vary widely.

Table 2: Rough comparison of CPU speeds by Clock Frequency Source: Providentia Worldwide illustrates a quick example using the sysbench tool for Linux of the differences between CPU models and manufacturers within the bounds of the same generation of processors and within similar clock frequency ranges. There is substantial variation such that assumptions which presume a roughly competent number of executed functions (SHA256 hashes in the case for Solana) could vary enough to cause concerns for both attack on the event streams alone or in combination, as well as brand risk with regards to platforms better suited for staked operations and higher neighborhood ranking.

Without specific tests performed using the Solana platform itself, it is difficult to understand the ease of exploit for these alternate platforms, but suffice it to say that the variability

[5] For example, the calculations and logic provided in sections 4.6, 4.7, and 6.3 (among others) in the Solana whitepaper all ignore a varied IO size when doing their analyses. This is fine to make benchmarking claims, but should be called out that uniformity is assumed for achieving these numbers.

[6] Due to the nature of the technology, the terms for “clock” and “time” can be overloaded. See the section for “conventions” for a deeper discussion of what we mean when these notions are referenced herein.

between platforms, even within those from the same processor vendor, are enough to warrant more serious and detailed investigation prior to making strong commitments and claims with regards to heterogeneity amongst supported processor platforms for the running code.

For a system to run optimized, an organization must calculate performance along generational boundaries, following calculations in Table 2. Additionally, controls and governance should be in place to ensure that there is a predictable march towards new capabilities as they arise, and new generations of behavior are well-understood before joining the network.

Processor	Clock Speed/ Architecture	*sysbench cpu --cpu-max-prime=20000 run* Events/sec
AMD EPYC 7401P	2.8Ghz / x64	526.84
Intel Xeon 5120	2.2Ghz / x64	427.75
Intel Core i5-8259U	2.3Ghz / x64	523.36
Qualcomm Centriq 2400	2.6Ghz / arm64	2349.71
Ampere eMag	3.3Ghz / arm64	536.72

Table 2: Rough comparison of CPU speeds by Clock Frequency Source: Providentia Worldwide

5.3 General Resource Limitations

As discussed above, the platform is not resilient to resource exhaustion and these edges have not been tested thoroughly. That is to say, that for any particular innovation, the required physical resources are generally presumed to be present, functional, and scalable without degradation or failure. While the overall platform does aim for scalability, the distinction for scaling along CAP[7] theorem lines is not well tested. Most components will show some unwanted behavior across the entire system when resources are exhausted. To compensate for this, operational controls which restart nodes, flush caches, or buffers should be considered.

For example, the Proof of History platform will grind to a halt if all leaders are unable to produce a new block. There are several ways in which this could occur, and they are discussed in the section below, but needless to say, this would result in the chain not making

[7] https://en.wikipedia.org/wiki/CAP_theorem. While generally well understood, the Solana team pointed out to the investigating team that they intentionally adopted the models for the Blockchain Trilemma as a means of looking at their platform. This is likely fine, but it tends to combine “availability” and “scalability” under the same banner. We recommend that an analysis of these individually will help them design a deployment architecture which does not have the same resource exhaustion constraints we observe in the discussion herein.

progress and would lead to unconfirmed transactions beyond the stated SLAs or worse, to a potentially forked chain altogether.

We recommend that all aspects of resource exhaustion, failure, and limitation be investigated by the Solana team and a full CAP analysis be undertaken to fully understand where the edges of the platform as implemented actually are and take action to adjust important gaps as part of the roadmap. Availability to process is considered a security risk, and as such handling situational failures, auto-restarting upon exhaustion, or other operational controls may compensate for these limitations.

5.4 Unverified Transaction Handling

The Gulfstream mechanism (6.5) implements a “distributed queue” by which unverified transactions will forward to the next leader for a slot until the block hash for the blockhash expires. This allows for normal back pressure to relieve and to handle spikes in transaction depth and volume. Once the block hash expires however, all unverified transactions must be handled by the client. There is no mechanism to enforce or govern behavior towards handling these failures in any particular client, however. This means that the default behavior would be to orphan transactions with no guarantee that they would be handled at all.

Any situation which would result in unconfirmed transactions should institute a process in which these transactions are eliminated or queued for further analysis. The current implementation relies on clients to handle this behavior, but without enforcement of the policy we see this as a risk of the implementation that should be better understood. For example, without client governance, valid retry attempts can result in effective denial-of-service and resource starvation. While this may be unavoidable, the precise behaviors should be understood and documented.

5.5 Verifiability of Functionality

Since the discussion with the team was largely around the current implementation and the testing conducted to date there are gaps between “planned functionality” and “implemented functionality” as well as between “tested implementation” and “current implementation”.

5.5.1 Planned Functionality

There are several innovations which are still under current development. As with any startup, this is expected behavior, but it means that for an analysis of this type, we must be able to draw a line between the behavior which is observable (implemented functionality) and that which is on the roadmap or under active development (planned functionality). In several cases, there are gaps where key pieces of the platform fall into the category of planned functionality. In the cases where security risk or vulnerability hinge on deploying planned functionality, we call that out.

Generally speaking, we do not list a finding which is planned unless there is no viable workaround in the implemented functionality or if the finding falls under a “High” classification and so could justify a change in the implementation strategy.

5.5.2 Tested Implementation

For the Solana platform, the day-to-day implementation and the tested implementations vary greatly in the number of validator/leader nodes, the number of spanned datacenters, the number and location of spanned cloud entities, the node configuration between providers, etc. Additionally, the team is well aware that the current implementation is not sufficient for long-term, global deployment at scale.

This presents a few problems for the evaluating team which are called out here:

- It is difficult to make a finding which results from this schism between current and future deployment. That is to say, some aspects of security which are directly related to scale may improve findings on smaller systems while others may exacerbate problems which may not exist on smaller deployments. Where this is the case, the report defers to the current, running platform and calls out differences.
- Since the scaling tests were measured using homogenous systems as much as possible, it is difficult to understand precisely how heterogeneity may play a role in availability and security concerns. When this happens, the report calls out that we suspect there may be some issues that arise due to scaling but we do not rank them higher than “Medium” severity since we cannot verify those claims.
- It is impossible to know precisely how all components interact at scale. Team-developed software is by nature different and behaves differently in the same situation. There are places where our assessment of growth may be spot-on and others which are nowhere close to our anticipations (both better and for worse). In these cases, we do our best to call out our suspicions as “Low” findings but cannot be responsible for things we miss altogether because our imaginations were not up to the task.

5.6 Cryptography Discussion

The Solana team had specific questions regarding the use of cryptography in the implementation. We summarize those questions here and refer to them again in their relevant sections under their innovation headings as well.

5.6.1 SHA-256 Hash Chains

SHA-256 is a good choice for the simple reason that it’s used in Bitcoin, so it has already been optimized, and it is reasonable to predict the computing time precisely for homogenous platforms. With enough testing across the realm of frequency and performance for different processors, the range of expected performance can be extended accordingly. It is important that tested code from known sources be used when implementing this algorithm.

5.6.2 CBC Chaining

Along with the hash chaining for SHA-256, this is a reasonable choice. See the discussion on quantum for details on elliptic curve.

5.6.3 Quantum-Proof Cryptography

The core principle for Solana is that the intended cryptography has "…has a 2^128 security target; breaking it has similar difficulty to breaking NIST P-256, RSA with ~3000-bit keys, strong 128-bit block ciphers, etc. The best attacks known actually cost more than 2^140 bit operations on average, and degrade quadratically in success probability as the number of bit operations drops.[8]" While there are "harder" curves, but they are not necessarily a better choice in terms of security (less studied) and performance.

With regards to quantum attacks, the system requires roughly 2330 logical qubits to break it. This translates to roughly 10M physical "today's" qubits. Notice that is roughly equivalent to quantum-breaking RSA-1024 (elliptic curve crypto is weaker than RSA from a quantum perspective).

As of writing of this paper, computers with approximately 53 qubits exist, but without the parallelization necessary to directly attack these algorithms.

5.7 Network Protocol Verification and Amplification

Cases where there are modifications to the network flow through the use of layer2 or layer3 devices result in unpredictable behavior between nodes and participants. There are a wide variety of these sorts of attacks, but a simple case is illustrated in Figure 4. In this example, additional traffic for turbine repairs are flooded from a malicious bridge device to all nodes. This traffic need not be valid to consume additional network resources as the protocol does not validate legitimate requests. The result can be an enormous increase in traffic on the wire and can effectively remove nodes altogether if the network bandwidth available to br0 exceeds that to any particular node under attack.

This vulnerability exists in the majority of the network protocols used for the Solana innovations, but mitigation should be possible. The particular implementation will determine the remediation method.

[8] See https://ed25519.cr.yp.to/ for a detailed description and continued conversation.

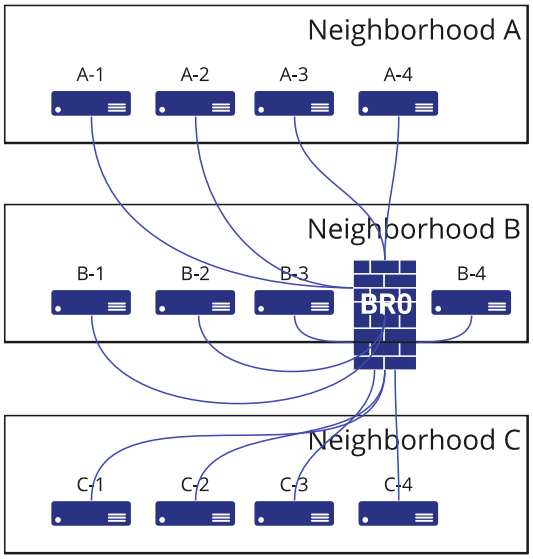


Figure 4: "Bump in the Wire" Amplification

6 Technical Details

6.1 Glossary

As with any new technology, there are a variety of acronyms and specific terminologies which are reserved or have special meanings for the Solana platform. We attempt to capture those relevant to our discussion here.

TERM	EXPLANATION
POH	Proof of History. A consensus mechanism which relies on the predictable passage of events (termed "*ticks*") which prove that data, transactions, or events attached to the blockchain happened prior to a specific *tick*.
BFT	Byzantine Fault Tolerance is a means for mitigating untoward behavior in a distributed system in which state in a distributed system cannot provide perfect information and so failures cannot always be fully understood, observed, or reacted to by other participants in the distributed system[9].
HORIZONTAL SCALING	A method for scaling parallel operations in a distributed system which accents atomicity and isolation. Generally, this is considered to be "scale out" where additional functional methods are added to the system in a fashion where cross-method state is not required[10].
PBFT	*Practical Byzantine Fault Tolerance*. The original paper located at http://pmg.csail.mit.edu/papers/osdi99.pdf assumes asynchronous communications between actors with consensus instead of mutual agreement at the core of the resolution. There are a variety of implementations of this consensus mechanism[11].
MEMPOOL	In common blockchain terminology, mempool refers to the current body of unconfirmed transactions awaiting block commits on the chain.
SIMD	*Single Instruction, Multiple Data* is the construct and implementation of parallel operations on data in vector computers and vector engines. This

[9] See https://en.wikipedia.org/wiki/Byzantine_fault for a good general discussion on the Byzantine Generals problem. See the Solana implementation discussion of their implementation for handling this behavior at https://solana.com/tower-bft/.

[10] See https://en.wikipedia.org/wiki/Scalability#HORIZONTAL-SCALING for a general representation, though the definition here is unclear as to where and how state might be managed. For the purposes of our discussion of Solana, we are looking particularly at the isolation of transactions between accounts to provide ordering and summation. Where decrements are required, the platform resorts to a serialized approach to guarantee atomicity with regards to account balances. See the discussion at https://solana.com/cloudbreak/.

[11] A good general discussion on the approaches leveraged in the blockchain community can be found at https://www.hyperledger.org/blog/2019/02/13/introduction-to-sawtooth-pbft. This is not an endorsement of the sawtooth implementation, but the logic of the discussion is sound and provides a good grounding for the underlying technical implementations shown here.

	approach applies single instructions, functions, methods, or kernels (depending on the implementation) to parallel data segments and is the general method provided by modern GPUs and multi-core CPU systems.
GPU	*Graphics Processing Unit*. A processing circuit implementation which leverages parallel data movement to perform vectorized operations. Originally designed for processing images through a frame buffer, these circuits are well tuned for parallel data operations and have grown in popularity for this purpose. See https://en.wikipedia.org/wiki/Graphics_processing_unit.
PROOF OF REPLICATION	*PoRep.* This is a Solana notion which provides notifications of block retention according to a defined replication scheme with provenance and fast verifiability.
TIME AND CLOCKS	See the section above for a discussion of conventions in use in this document. Since time and clocks are key to the technology evaluation and are used in a variety of ways, this bears a longer exploration than can be given in a glossary entry.
CAP	*Consistency, Availability, Partition Tolerance*. A theoretical computer science and engineering framework to understand how any computer system reacts to certain behaviors and failures and what the tradeoffs and impacts of those might be.
SHRED	In turbine, a shred is a chunk of an overall block which is being distributed for verification across neighborhoods. Each block is shredded into a number of components, each of which is signed and sent to a shuffled participant list.

Table 3: Glossary

6.2 Innovation: Proof of History (POH)

6.2.1 General Description

The Proof of History is a high frequency Verifiable Delay Function[12]*. 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. [The Solana] specific implementation uses a sequential pre-image resistant [SHA256] hash that runs over itself continuously with the previous output used as the next input. Periodically the count and the current output are recorded [(ticks)]. For a SHA256 hash function, this process is impossible to parallelize without a brute force attack using* 2^{128} *cores. We can then be certain that real time has passed*

[12] The original reference from Solana points to the following Stanford talk: https://www.youtube.com/watch?v=qUoagL7OZ1k.

between each counter as it was generated, and that the recorded order each counter is the same as it was in real time.

Proof of History represents a sea-change for backing consensus algorithms for the blockchain ecosystem. Its core aim is to enhance the transactional speed of block verification and thereby provide trust in a way which can be leveraged efficiently and provably in consensus, verification, ordering, and a variety of other common computer engineering tasks. It would function admirably as a standalone service (oracle) as well.

6.2.2 Security Claim Findings

6.2.2.1 Clock Frequency

As mentioned in the section above, Proof of History is perhaps the most vulnerable to resource variability between clock frequencies on the deployed systems. That is to say that while the theoretical claims about the functionality for Proof of History generally follow according to their structure, the detailed implementation may vary widely. As in the table presented above, clock frequency cannot be taken as the sole means by which a verifiable delay has happened and certainly cannot be counted on to present the same amount of elapsed time in all cases.

We should to be able to better understand the actual impact of heterogenous deployments when it comes to staking leaders (see TowerBFT). As an independent PoH deployment – such as a blockchain oracle providing a Proof of History tick stream – the design and implementation are not problematic, but when coupled with varying leader configurations the actual elapsed time may not prove as predictable as claimed. In and of itself this is not necessarily a problem with it as written, but it may mean that in certain situations a predictable transaction rate may not be able to be sustained if there are wide variations in the number of text and the frequency of two changes between leaders.

Nevertheless, even for usage as an oracle, the implications of resource variability for generating ticks should be understood and documented.

6.2.2.2 Vector Clocks

The Solana whitepaper outlines a section detailing an implementation of multiple PoH clocks interleaving their clock and transaction streams. This is an interesting area and under current development in computer science[13]. However, the Solana team has made the decision not to investigate vector clocks at this time and so our evaluation does not go into detail here besides capturing the theory in the whitepaper.

[13] See https://en.m.wikipedia.org/wiki/Vector_clock for a discussion of the concept. The whitepaper discussion shows essentially the same behavior here. It is important to note that any findings which are specific to PoH itself would be inherited in a vector implementation of multiple clocks and could result in additional findings beyond simply compounding understood vulnerabilities. The Solana team recognizes that this is a complex branch of computer science however, and so has decided not to pursue this at this time. We therefore also choose not to delve deeply into this area either as there is no implemented functionality in the current platform.

6.2.3 Innovation Dimension Ranking

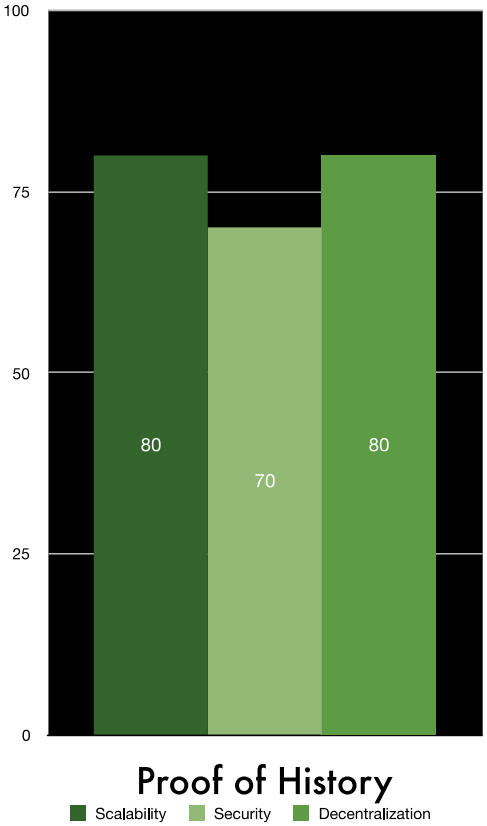


Figure 5: IDR Proof of History

For full Scalability marks, Solana should investigate integrating vector clocks to enable horizontal scaling of clock sources.

Decentralization needs to be proven at production, full performance scale with heterogeneous participants and true permission-less attributes.

6.3 Innovation: Tower BFT

6.3.1 General Description

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 time-outs that replicas use in PBFT can be computed and enforced in the PoH itself.

PoH is a Verifiable Delay Function implemented as a sequential hash function. … The basic principles of how PoH works are as follows:

1. *Sha256 loops as fast as possible, such that each output is the next input.*

2. *The loop is sampled, and the number of iterations and state are recorded.*

The recorded samples represent the passage of time encoded as a verifiable data structure. In addition, this loop can be used to record events.

1. *Messages that reference any of the samples are guaranteed to have been created after the sample.*
2. *Messages can be inserted into the loop and hashed together with the state. This guarantees that a message was created before the next insert.*

This data structure guarantees both time and order of events embedded within, and this core idea is the basis of all of the major technical optimizations in Solana.

6.3.2 Security Claim Findings

6.3.2.1 Voting Assumptions

Generally, PBFT requires that all participants actively participate in the network. This is fundamental to the "practical" aspects of BFT since it is theoretically possible for a general to choose not to fight at all under any circumstances. In traditional BFT, this results in a smaller pool of attackers and could change the outcome or success of consensus if enough generals simply choose not to vote.

In TowerBFT, voting is constant and tied to "staking" a leader pool with known validators and predictable tick rates. However, in a collusion event, it would be possible to cause a network partition simply by having preferred leaders (those known to be coming next in the election pool) choose not to participate for enough time that forced elections to occur. With enough collaborators, a forced fork could happen. The mitigation strategy of "slashing stakes" is meant to disincentivize participants such that they always vote (and tend to vote the vast majority of the time on the consensus block).

There is no enforcing logic for this however, so if colluding leaders decide simply not to vote, there is nothing which will entail the network to make progress. This could be an existential risk to the core chain itself and could result in a variety of outcomes including:

- Unverified transactions
- Dropped transactions
- Chain forks
- Currency devaluation/valuation
- Resource denial or redirection to other nodes. (For example, forcing remaining "good leaders" to take on the bulk of work until they become resource constrained or overloaded and effectively take themselves out of the network causing a complete standstill.)

While many of these states are known and understood, and not specifically particular to TowerBFT's behavior, we call them out here since each needs to be understood, analyzed, and mitigated (or consciously decided not to be mitigated) individually. While the team has spent a great deal of effort in managing and planning how to handle these situations, very few of them have actually been tested and so the behavior and outcomes remain theoretical. As a result, we call this out as an additional risk and so elevate the finding level. A recommendation during lack of voting and scenarios discussed above should be an item handled within the code itself, possibly allowing the system to gracefully recover under such a scenario.

6.3.2.2 Network Partitioning

Many of the scenarios discussed in the analysis of TowerBFT centered on the ability to partition and attempt to take control of the voting majority of a partition or re-assembly. This is deemed as difficult and generally requires collusion between the clock source and the leader. However, in the current implementation, clock source and the leader pool are composed of the same systems and so we find that this collusion would be the usual case and not the exception.

There has not been a great deal of network partitioning testing done beyond the usual failure scenarios that occur during development. As a result, there has also not been a great deal of implementation effort placed in shortcut recovery or specific-scenario handling.

Because not all scenarios are tested[14] and the current implementation is not at scale (5 nodes see Figure 6), we cannot provide details on the engineering design to carry out particular partition scenarios. We will elucidate the different categories here as figures for ease of reference in the discussion following. We will use the number of participating validators in the pool at the time of writing to elucidate these scenarios. While there are tests conducted at a variety of larger scales, all are under controlled circumstances and do not always reflect the behaviors of long-term production workloads or show behavior under adverse conditions. This is not meant to disparage or discourage continued testing however, just to point out that we need to better understand behaviors in these environments[15].

[14] There are many tests available and more changing all the time. We anticipate that many of the scenarios described here will be added and updated over time. For Solana's ongoing efforts with testing and analysis see https://github.com/solana-labs/solana/blob/v0.20.0/local_cluster/src/tests/local_cluster.rs and for TowerBFT partition simulations see: https://github.com/solana-labs/solana/blob/v0.20.0/core/tests/fork-selection.rs. For an example of testing with 206 nodes, see this video: https://www.youtube.com/watch?time_continue=1&v=BG2AHQtHysA

[15] For additional conversation regarding the size of tests and deployments for Solana, please refer to 2.1 above. Likewise, Solana is continuously updating their covered testing and will likely have already made adjustments by the time this document is published.

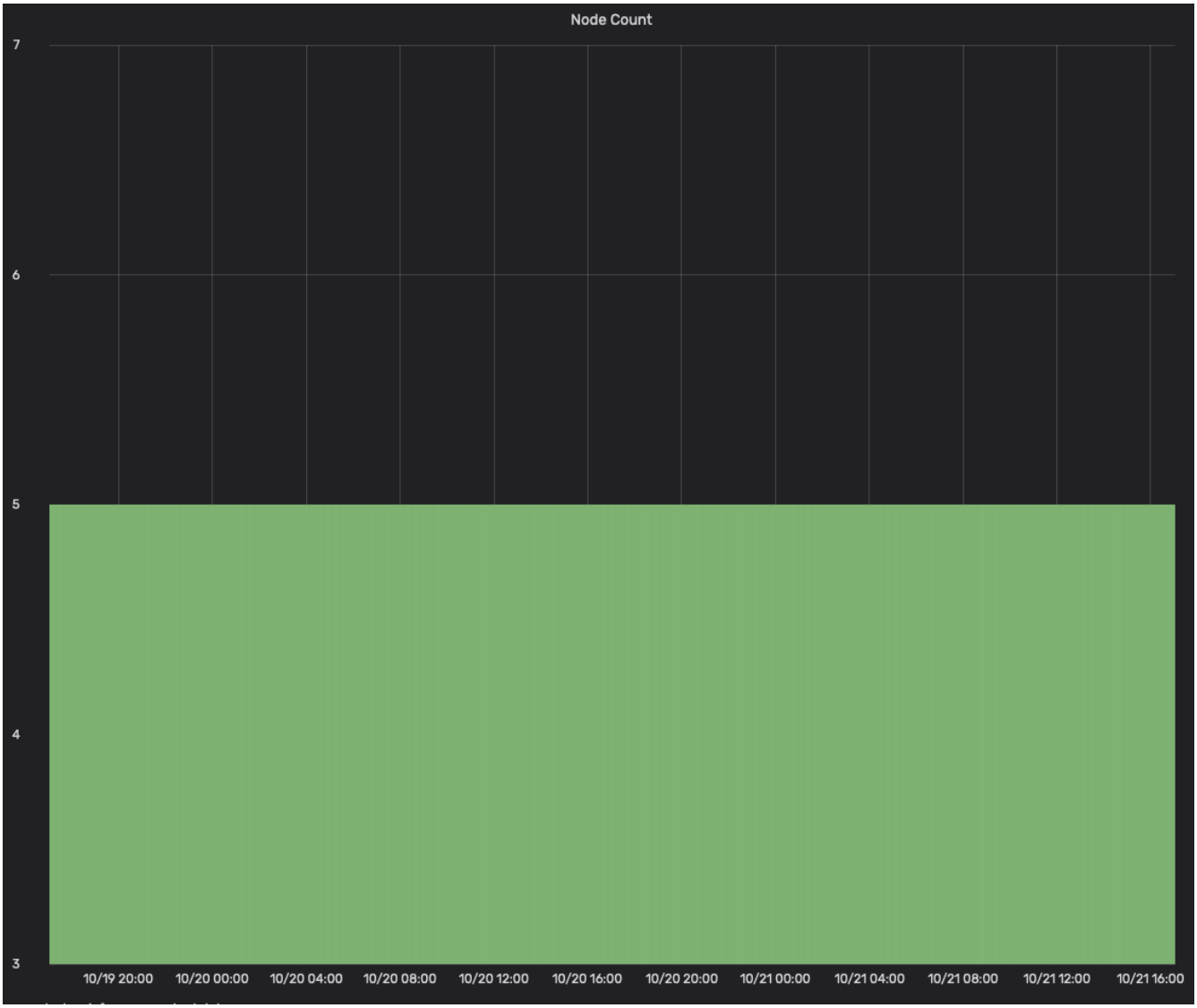


Figure 6: Solana Testnet Node Count (at time of document creation)

6.3.2.2.1 Network Partition Scenarios

This is not an exhaustive list of the potential areas for the network to partition, but each aims to capture an aspect of the failure which must be understood and mitigated for production deployments – especially in a permission-less network. We have tried to present these in order of compounding findings to allow for an additive approach to describing the scenario and potential impact.

As there are several ways in which network partitions occur, we capture the key categories in Table 4 to reference behavior as we discuss each scenario.

Table 4: Network Partition Failure Types

Failure ID	Failure	Description
1	Missing Node	A node disappears from the network. There is no termination of work or clean cessation. This is the

		equivalent of unplugging the network cable. The impacted node may or may not be available locally.
2	Disconnected Node	A node terminates connections and activity but is still accessible on the network for other protocols (ping, etc.). In this case the node may or may not be available locally and may or may not be available remotely through other means. For the purposes of network participation, this node is refusing communications.
3.1	Intermittent Node	A node is suffering from failure ID 1 intermittently. The interval may be random or predictable. An example would be a system with a shorting cable which occasionally provides connectivity or a node suffering from spanning tree convergence and is alternating from blocked or forwarding.
3.2	Intermittent Node	A node suffering from failure ID 2 intermittently. The interval may be random or predictable. An example would be a node starting the leader process and failing recurrently.
4	Partial forwarding	A node which is either sending or receiving traffic successfully but not both. This could be implemented by bridge or firewall rules depending on the forwarding layer.
5	Administrative Partitioning	A node which is prohibited from exchanging data with some portion of the network through external (to the validator network) means. Examples would include a firewall prohibiting node A from communicating with node E but allowing other communication, or a node A which is unable to reach node E due to temporary or permanent routing failures.
6	Assumed Identity	A node which intentionally behaves as another participating node. This scenario could be the result of a split-brained virtual machine or container migration resulting in two nodes functioning as node A.

6.3.2.2.1.1 A: Single Divide

Figure 7 illustrates a simple network partition where sides 1 and 2 represent the partitions respectively. In this case there are is a majority of staked leaders in partition 1 and the presumed behavior is that these leaders will quickly unstake nodes D and E and thereby carry on the fork even after partition reassembly. Also presumably, partition 2 will be increasingly unlikely to obtain consensus since it is trying to reconstruct a minimum-3 node quorum and cannot succeed before the increasing block validation timeout essentially reaches infinity (for Solana this number is set to 32 which equates to roughly 54 years).

It is not clear that the team has tested failure ID 6.

For a colluding partition 1, the fork would result in chain governance remaining with partition 1. For a colluding partition 2, there would not be agreed consensus without a new genesis and leader staking.

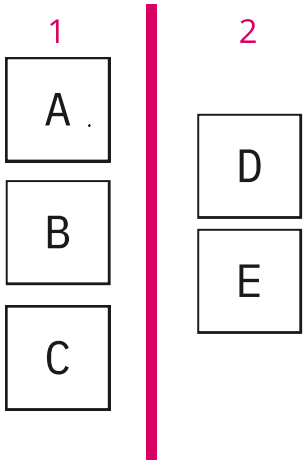


Figure 7: Network Partition A, Single Divide

6.3.2.2.1.2 B: Multiple Divide

Figure 8 illustrates a multiple network partition where sides 1, 2, and 3 represent the partitions respectively. All three partitions result in minority stakes and cannot make progress towards staked convergence.

This scenario has not been tested under the failure matrix (Table 4).

It is unclear what behavior would occur in colluding networks in this scenario. It is also unclear if it is possible to enact failure scenario ID 6 in this model, but it should be designed and tested if possible.

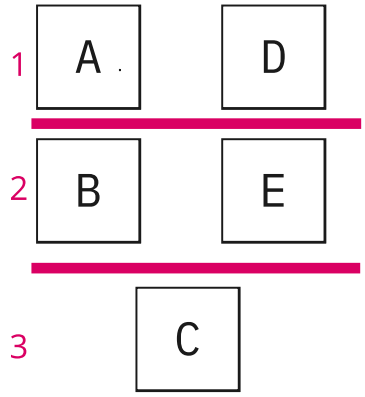


Figure 8: Network Partition B, Multiple Divide

6.3.2.2.1.3 C: Single Divide, Secondary Divide

Figure 9 illustrates a multiple network partition where sides 1, 2, and 3 represent the partitions respectively. In this scenario the original left-right division has a subdivision after elections and staking/slashing begins resulting in a new subdivision. It is unclear what the behavior would be if the network had converged in favor of A (Scenario A) or if it was still constructing.

This scenario has not been tested under the failure matrix (Table 4).

It is unclear what behavior would occur in colluding networks in this scenario. It is also unclear if it is possible to enact failure scenario ID 6 in this model, but it should be designed and tested if possible.

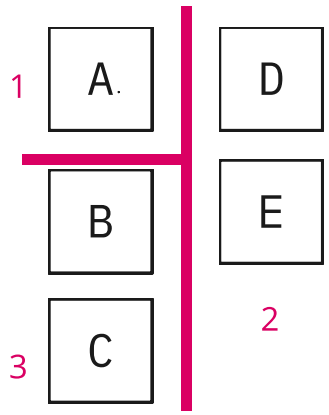


Figure 9: Network Partition C, Single Divide, Secondary Divide

6.3.2.2.1.4 D: Single Divide, Multiple Secondary Divide

Figure 10 illustrates a multiple network partition where sides 1, 2, 3, and 4 represent the partitions respectively. In this scenario both original divisions have a subdivision after elections and staking/slashing begins resulting in a new subdivision. The behavior of this type of partitioning is unclear.

This scenario has not been tested under the failure matrix (Table 4).

It is unclear what behavior would occur in colluding networks in this scenario. It is also unclear if it is possible to enact failure scenario ID 6 in this model, but it should be designed and tested if possible.

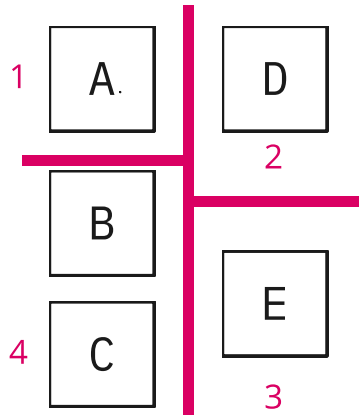


Figure 10: Network Partition D, Single Divide, Multiple Secondary Divide

6.3.2.2.1.5 E: Single Divide, Reassembly Competition

Figure 11 illustrates a multiple network partition where sides 1, 2, and 3 represent the partitions respectively. In this scenario, there is competition on reassembly between nodes BC and DE as they have respectively slashed one another prior to network reassembly. The behavior of this type of partitioning is unclear. It is particularly unclear the role that a presumably excluded/slashed node A would take in network convergence.

This scenario has not been tested under the failure matrix (Table 4).

It is unclear what behavior would occur in colluding networks in this scenario. It is also unclear if it is possible to enact failure scenario ID 6 in this model, but it should be designed and tested if possible.

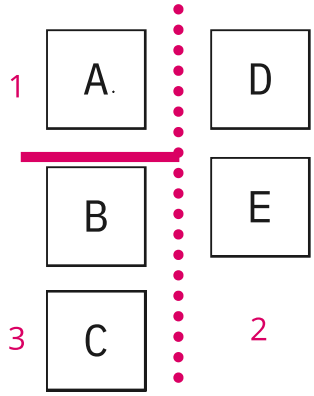


Figure 11: Network Partition E, Single Divide, Reassembly Competition

6.3.2.2.1.6 F: Permanent Divide, Reassembly Competition

Figure 12 illustrates a scenario much like scenario E where the original division is permanently split. This could be due to an unresolvable recovery condition (network connectivity loss for example) or through an intentional (malicious or benign) fork of the network. Reassembly competition occurs between nodes AB and C in division ABC1 and between DE in division DE2.

This scenario has not been tested under the failure matrix (Table 4).

It is unclear what behavior would occur in colluding networks in this scenario. It is also unclear if it is possible to enact failure scenario ID 6 in this model, but it should be designed and tested if possible.

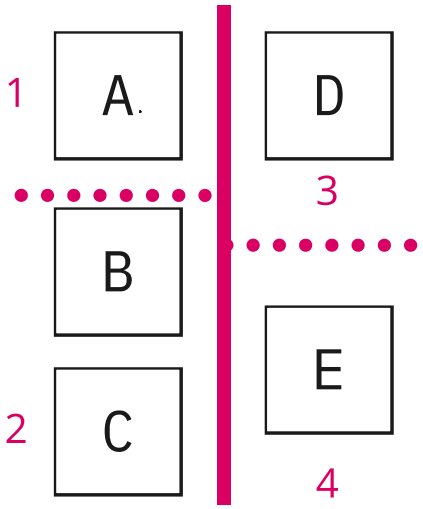


Figure 12: Network Partition F, Permanent Divide, Reassembly Competition

6.3.2.2.1.7 G: Single Divide, Minority Intentional Participation Exit

Figure 13 shows the case of an intentional fork where partition 2 decides to exit the overall network and continue on a new fork. This is a minority stake intentionally preserving a chain on their own. While the reassembly and continuation of the larger staked chain is understood here, what is particularly unclear is what the behavior of clients might be. In a colluding situation where nodes DE claim to be majority stakeholders (even though they are not), what happens to their clients and associated transactions?

This scenario has not been tested under the failure matrix (Table 4).

It is unclear what behavior would occur in colluding networks in this scenario. It is also unclear if it is possible to enact failure scenario ID 6 in this model, but it should be designed and tested if possible.

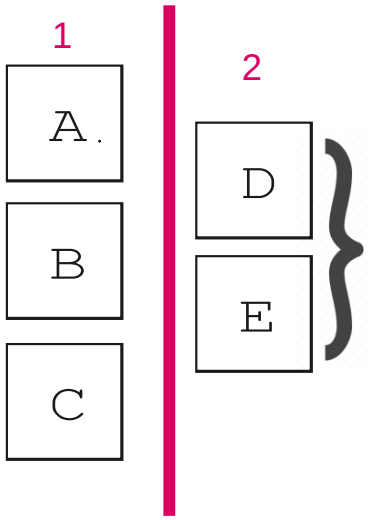


Figure 13: Network Partition G, Single Divide, Minority Intentional Participation Exit

6.3.2.2.1.8 H: Single Divide, Majority Intentional Participation Exit

Figure 14 shows the case of an intentional fork where partition 1 decides to exit the overall network and continue on a new fork. This is a majority stake intentionally preserving a chain on their own. While the reassembly and continuation of the larger staked chain is understood here, what is particularly unclear is what the behavior of clients might be. Presumably this is a standard case for an intentional fork, but the nature of how colluding malicious nodes might carry off this attack needs to be fleshed out and understood.

This scenario has not been tested under the failure matrix (Table 4).

It is unclear what behavior would occur in colluding networks in this scenario. It is also unclear if it is possible to enact failure scenario ID 6 in this model, but it should be designed and tested if possible.

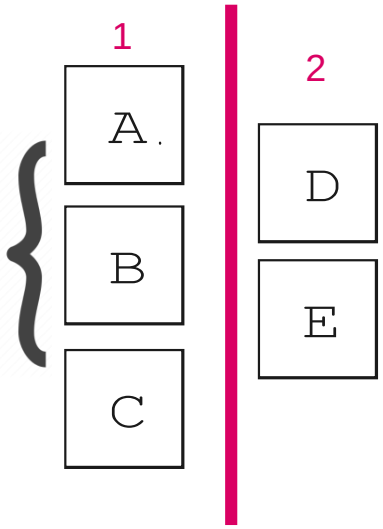


Figure 14: Network Partition H, Single Divide, Majority Intentional Participation Exit

6.3.2.3 SHA-256 Hash Chains

The Proof of History mechanism as implemented in TowerBFT relies on SHA-256 hashing. See the general discussion (5.6.1) for context and explanations here.

6.3.3 Innovation Dimension Ranking

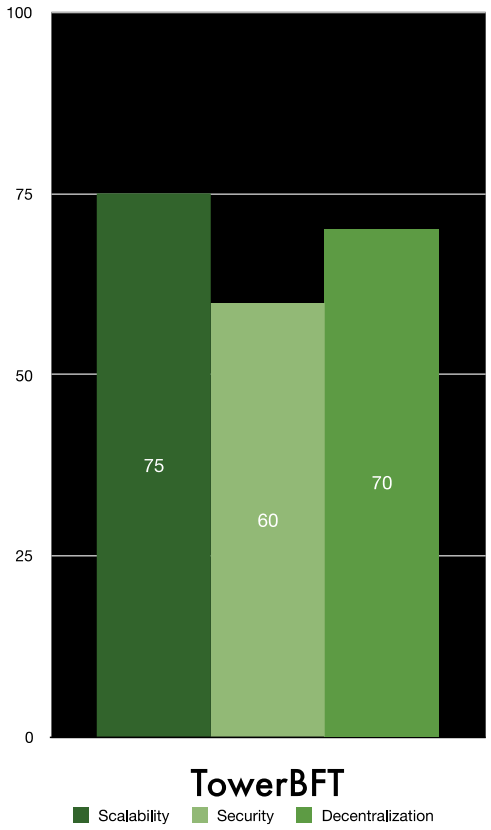


Figure 15: IDR TowerBFT

All dimensions need to be proven at full production scale with permission-less governance and in heterogeneous computing environments.

6.4 Innovation: Turbine

6.4.1 General Description

One of the challenges to high-performance blockchains is how the network propagates large amounts of data to a large number of peers. [Solana's approach] to this problem, Turbine, borrows heavily from BitTorrent — although a few major technical details differentiate the two. Turbine is optimized for streaming, and transmits data using UDP only, implements a random path per packet through the network as leaders (block producers) stream their data. The leader breaks the block into packets up to 64KB in size. For a 128MB block, the leader produces 2,000 64KB packets, and transmits each packet to a different validator.

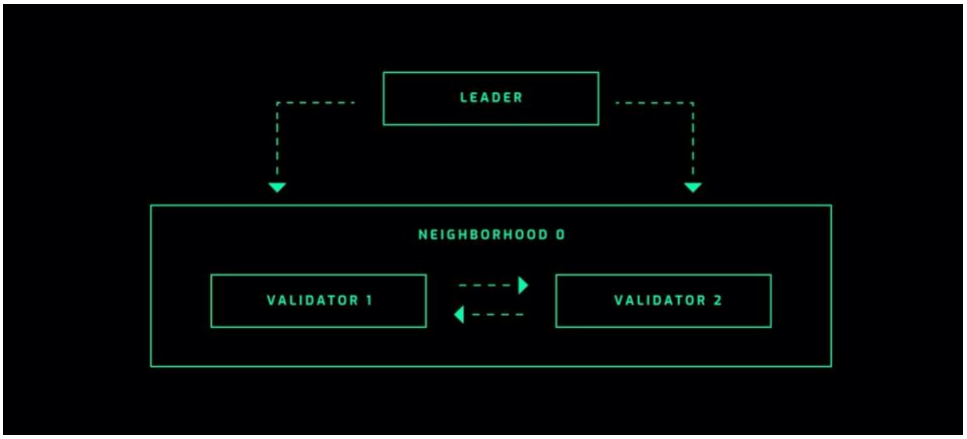


Figure 16: Turbine neighborhood 0. Source: Solana

In turn, each validator retransmits the packet to a group of peers that we call a neighborhood.

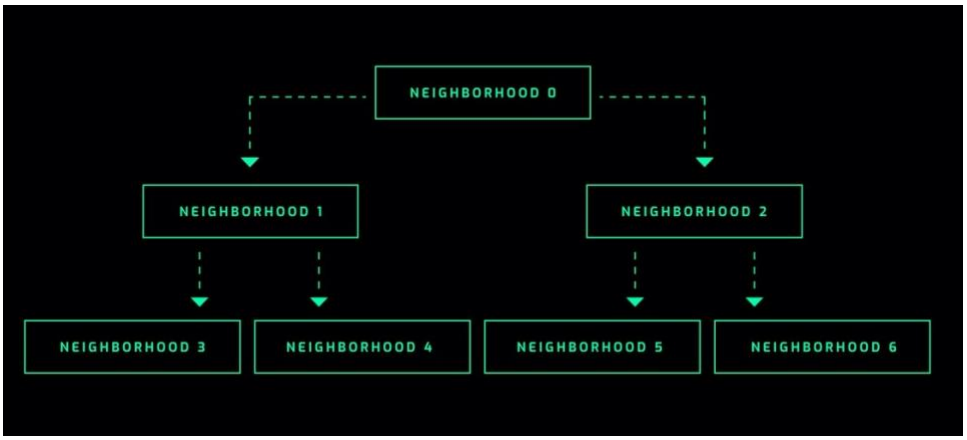


Figure 17: Turbine neighborhood tiers. Source: Solana

Each neighborhood is responsible for transmitting a portion of its data to each neighborhood below it.

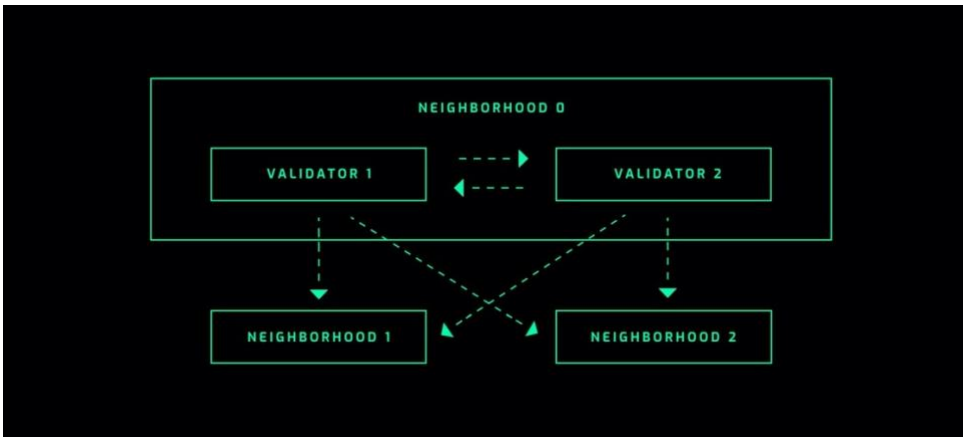


Figure 18: Turbine data distribution. Source: Solana

If each neighborhood is comprised of 200 nodes, a 3-level network, starting with a single leader at the root, can reach 40,000 validators in 2 hops — or roughly 200 milliseconds

assuming each network link is 100ms on average. … To handle adversarial nodes, the leader generates Reed-Solomon erasure codes[16] [allowing] each validator to reconstruct the entire block without receiving all the packets.

… A stake-weighted selection algorithm constructs the tree such that the higher staked validators are at neighborhoods closer to the leader. Each validator independently computes the same tree. … [The Solana] fanout algorithm generates a stake-weighted tree for every packet using a random source based on the digital signature of the packet. Since each packet takes a different path, and the path is not known in advance, a neighborhood-level eclipse attack would require nearly full control of the network.

6.4.2 Security Claim Findings

6.4.2.1 Network Distribution

Turbine allows any participating node to request blocks from any other node in its list. This means that it willingly accepts blocks from anyone who is in the nodelist at all. This behavior, while useful for recovery operations, allows a variety of attacks. These may include:

6.4.2.1.1 Bandwidth Overruns

This is the case where the available bandwidth for a particular node or neighborhood may be over-extended through responses for required blocks. While the initial distribution method per block does its best to ensure that there is only single delivery for blocks, when repair requests are made it is easy to overrun the available bandwidth on responses. This is exacerbated in cases where the network behavior is modified such that alternate paths are not available, routing and/or spanning loops exist, amplification attacks are employed, or network partitions exist between responding and requesting nodes.

This is a protocol-specific failure scenario in the same vein as ID 5.7 (Network Protocol Verification and Amplification).

6.4.2.1.2 Unidirectional Traffic Isolation

In the event of controlled or blocked network flow, Turbine participants affected by the impairment are unable to retrieve complete blocks. Since there is no facility beyond gossip announcements however, the requests for repair alone can cause a cascade of requests both within and between neighborhoods. Turbine does not authenticate or validate requests as they come in from other nodes. This means that participants will hear requests and respond, but blocked flow will prevent ever receiving the block, resulting in another request, and so on until the network amplification cascades. In Figure 19, node B-4 can only make outbound requests and can therefore never receive a block. All of the other participants will get the repair requests and attempt to respond regardless. Even in cases where the system rate-limits repairs, an attacker need only ignore this behavior and send anyway since there is no

[16] The original source on the Solana blog references this discussion: http://smahesh.com/blog/2012/07/01/dummies-guide-to-erasure-coding/.

input verification or validation for content or request validity. Since Turbine uses UDP, it relies on the implementation to handle these failures efficiently.

This is a protocol-specific failure scenario in the same vein as ID 5.7 (Network Protocol Verification and Amplification).

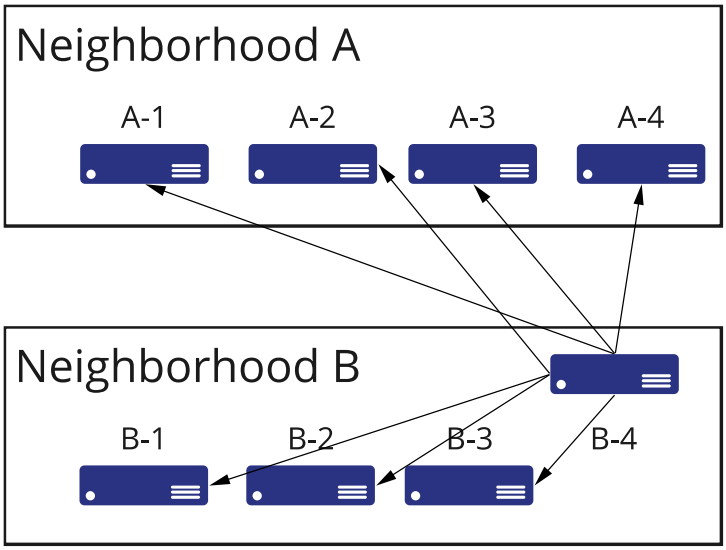


Figure 19: Unidirectional Flow

6.4.2.1.3 Network Partition: Failure Partitions

In these partition scenarios, a neighborhood loses connectivity between participating neighborhood nodes (Figure 20) and between neighborhoods (Figure 21). In these cases, normal repair requests should reconstruct needed blocks quickly. These failure scenarios have not been tested under permission-less conditions or between heterogenous platforms with wide latency boundaries between participating and affected nodes.

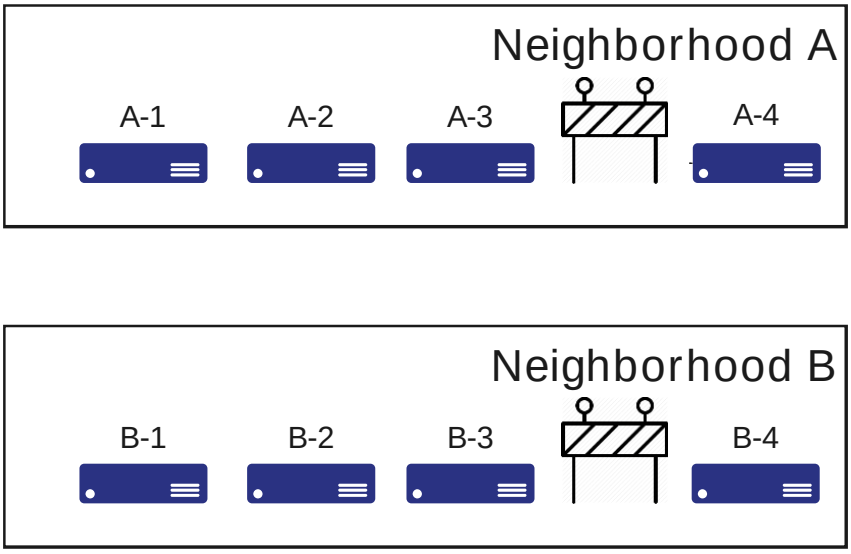


Figure 20: Network Failure Partition A, Neighbors

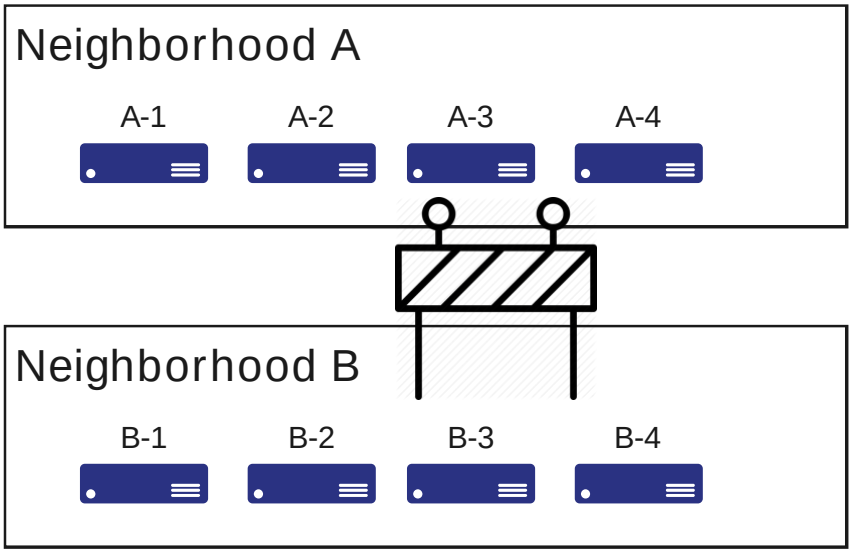


Figure 21: Network Failure Partition B, Neighborhoods

6.4.2.1.4 Network Partition: Administrative Partitions

This is a special case of network failure partitions where for administrative reasons some portion of connectivity between nodes in the node list (within or between neighborhoods) is prevented. For example, in Figure 22 if Node B-3 is located in New York, USA and node B-4 is located in Pyongyang, North Korea, it is currently illegal for any electronic communication to pass between those nodes regardless of purpose. A firewall prevents this communication. However, since the node list is passed along with the block, node A-1 and A-3 will be informed of each other's existence and methods by which connections can be made. There is no

facility in Turbine to prevent communication between these nodes however, and so there will always be a connectivity failure result. Likewise, during these windows, spoofing and man-in-the-middle style compromises may be possible due to the inability for communications to pass.

While the situations above can explain how the system would handle this, in a large, permission-less distributed implementation of the Turbine system, this behavior could occur constantly between many nodes in the system and result in unpredictable behavior with regards to block exchange. We suggest that much testing in this area be conducted to understand, anticipate, and mitigate failure and untoward scenarios.

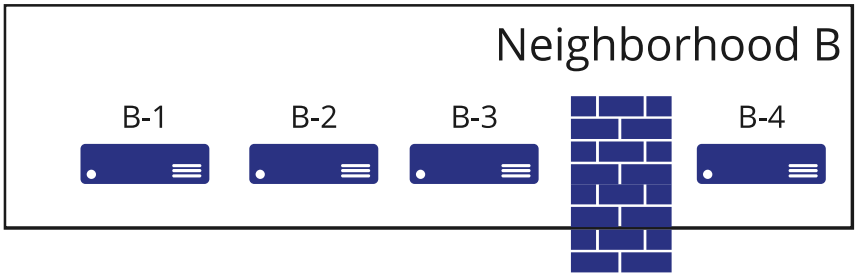


Figure 22: Network Partition, Administrative Prohibition

6.4.3 Innovation Dimension Rankings

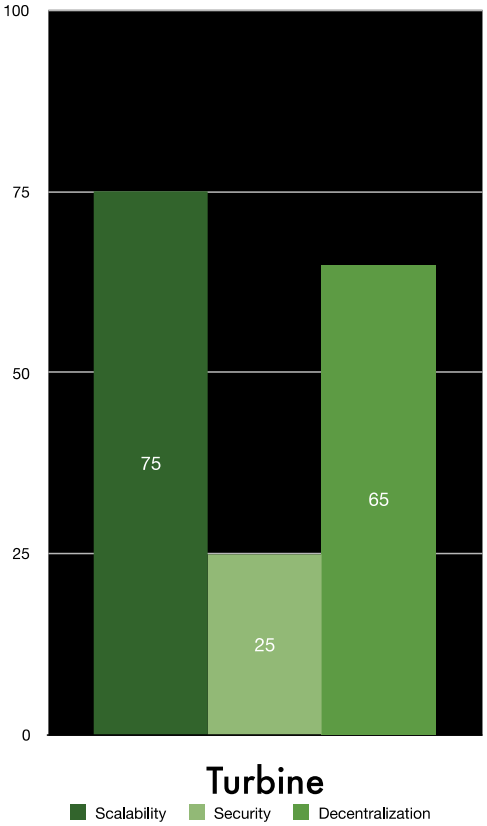


Figure 23: IDR Turbine

Scalability:

The platform needs to

1. ensure that authentication for transfers is in place;
2. reasonable limits on the number of transfers, transfer size, re-transmissions, etc. exist;
3. that there is internet-best practice communications security in place;
4. that performance penalties for introducing wire-line security measures are handled and adjusted accordingly.

Security:

The innovation is designed and implemented for performance first, and so does not have many of the needed security systems in place to be leveraged on a permission-less, public system.

Decentralization:

Because of the needs for both scalability and security, the system does not lend itself to acceptable decentralization at this time and should only be run on a set of controlled nodes.

6.5 Innovation: Gulfstream

6.5.1 General Description

A mempool is a set of transactions that have been submitted, but have not yet been processed by the network. … Solana validators … push transaction caching and forwarding to the edge of the network. Since every validator knows the order of upcoming leaders, clients and validators forward transactions to the expected leader ahead of time. This allows validators to execute transactions ahead of time, reduce confirmation times, switch leaders faster, and reduce the memory pressure on validators from the unconfirmed transaction pool. This solution is not possible in networks that have a non-deterministic leader.

So how does it work? Clients, such as wallets, sign transactions that reference a specific block-hash. Clients select a fairly recent block-hash that has been fully confirmed by the network. … Once a transaction is forwarded to any validator, the validator forwards it to one of the upcoming leaders. Clients can subscribe to transaction confirmations from validators. Clients know that a block-hash expires in a finite period of time, or the transaction is confirmed by the network. This allows clients to sign transactions that are guaranteed to execute or fail.

6.5.2 Security Claim Findings

6.5.2.1 "Fast Forward"

This is the notion that we can attack with collaboration between an upcoming validator and the known election process to inject many transactions into the next block when our turn comes around in such a manner that causes delays or failures in verification and results in a network partition. For example, in the figure below, if node C pre-calculates 1 billion transactions (or so many transactions that it takes a long period to verify/replay those transactions) while it is continuing its function as validator, only to insert all 1 billion instantly when its leader slot arrives, then it could cause delays, failures, or resource constraints on A and B such that they are unable to move forward, crash, or suffer other detrimental behavior.

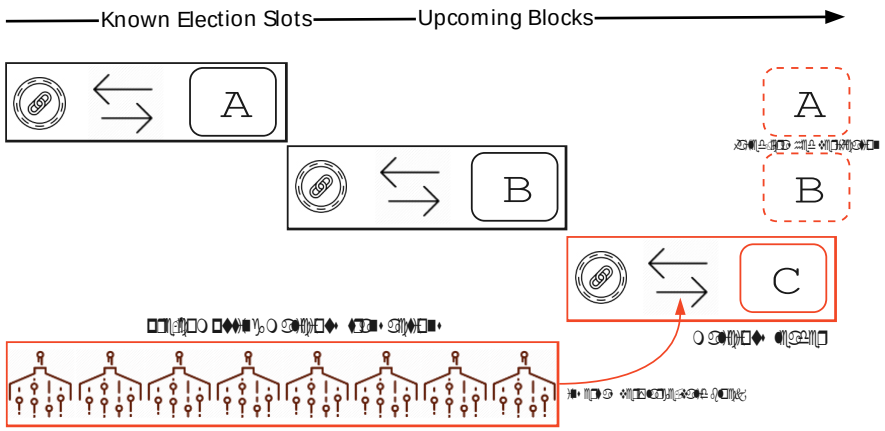


Figure 24: "Fast Forward"

6.5.2.2 *"Slow to a Crawl"*

A variance of the Fast Forward attack where the intention is to continue to validate extremely large transactions such that the network does not make progress. Transactions will continue to come in from clients, but there is no means for consensus to be reached. This requires collusion between multiple potential validators and known leaders. We detail this attack more under the section for Proof of History since this is technically possible at all implementations of PoH or TowerBFT.

6.5.2.3 *"Unverified Masses"*

Since the mempool structure of Gulfstream relies on the effective distribution of accounts (see Cloudbreak), then it appears that transactions which create many accounts could results in active account abandonment between transactions and eventual resource exhaustion and partitioning. It is not clear where the resource edges actually lie, but contenders could be

- Filesystem buffer cache
- File to memory pinning
- Disk IO for file creation
- Disk contention for active account thrashing
- Barrier contention for account processing (see Pipelining).

This area should be thoroughly tested to understand the behaviors here at scale and get better best-practice recommendations for any developed findings in this area.

6.5.3 Innovation Dimension Ranking

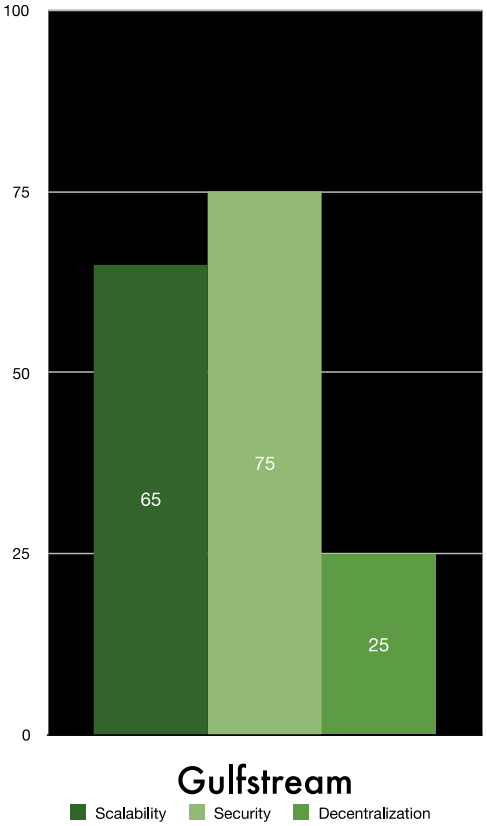


Figure 25: IDR Gulfstream

Both scalability and decentralization need to be proven at scale with heterogenous participants. It is unclear how this will impact memory sizes, network paths, and the like or if those changes will impact the forwarding behavior and the amount/speed at which transactions can be forwarded. Since this is an internal improvement, the “lowest common denominator” problem (2.2) is prevalent here.

6.6 Innovation: Sealevel

6.6.1 General Description

Solana’s parallel smart contracts runtime … can process tens of thousands of contracts in parallel, using as many cores as are available to the Validator. [Since] Solana transactions describe all the states a transaction will read or write while executing … non-overlapping transactions [can be permitted] to execute concurrently, [as well as] transactions that are only reading the same state to execute concurrently.

[Solana’s] key insight here is that programs are code, and within [their] key-value store, there exists some subset of keys that the program and only that program has write access to. Transactions specify an instruction vector. Each instruction contains the program, program

instruction, and a list of accounts the transaction wants to read and write. This allows [the Sealevel runtime] to prefetch, prepare the device, and execute the operation concurrently if the device allows it. On Solana, each instruction tells the VM which accounts it wants to read and write to ahead of time. This is the root of our optimizations to the VM.

1. *Sort millions of pending transactions.*
2. *Schedule all the non-overlapping transactions in parallel.*

SIMD[17] instructions allow for a single piece of code to execute over multiple data streams. This means that Sealevel can execute an additional optimization, which is unique to Solana design:

1. *Sort all the instructions by program ID.*
2. *Run the same Program over all accounts concurrently.*

The [Solana Sealevel] multiprocessor is bound by the slowest path that execution will take in the batch.

6.6.2 Security Claim Findings

6.6.2.1 Unverified Rust Implementation of eBPF

The team implemented their own interfaces into the Linux extended Berkeley Packet Filter structure. While the BPF technology is well understood and proven, there has not been a great deal of oversight or analysis into the Solana implementation directly. We recommend a thorough code analysis and testing framework for the implementation generally and looking into the intellectual property ramifications of the implementation. The implementation is currently open sourced at the following URL: https://github.com/solana-labs/rbpf.

6.6.2.2 Account Management

The current account management and allocation mechanism is implemented on-chain using the Sealevel engine. The methodology for the engine is handled in other sections, but if there are implementation bugs and issues related either to the Rust implementation of the eBPF interface then those would roll up under this set of concerns.

6.6.2.3 Auditability

For secure operations, there needs to be an audit trail of software execution and behavior. The current eBPF implementation does not allow for either on-chain or external auditing. When asked to the core team, this is a planned extension of the Sealevel engine but there is no determined timeframe on delivery of the functionality at this time. Auditability of financial

[17] The original discussion references https://en.wikipedia.org/wiki/SIMD to discuss SIMD operations. As this is primarily a performance improvement, the discussion on Sealevel in this document a limited to the security implications of potential behavior changes and malicious code attempts. There are a variety of additional implications however, and the reader is encouraged to consider the ramifications of these in their own investigations.

transactions would likely be required for organizations using this type of system to control movement of financial instruments or stores of value.

No discussion on the method for implementation was had at the time.

6.6.2.4 De-Optimizer

There are essentially 3 key optimizations in Sealevel which implement parallelization of incoming transactions by grouping them into batches and enhancing throughput. These are:

1) All transactions which do not have interdependencies are sorted into batches which can be processed in parallel (in the diagram below these are A>B, C>D, E>F);
2) Batches with multiple credits and no debits to the same account are optimized as a single operation (A>B, C>D, but not E>F as there is only a single transaction);
3) Anything which does not fall into optimizations 1 or 2 (which means anything with shared data dependencies) are single-threaded to maintain accounting principles and ordering of transactions.

Since the logic for Sealevel parallelization ensures that any operation against an account in a batch of transactions which is not a credit results in a single-threaded behavior for that batch, malicious attacks which force debits for all accounts would result in a slowdown for all transaction processing. For example, in the figure below, we have an Evil account which is well-funded send millions of continuous credit transactions to all accounts. This will force all transactions to become single-threaded (save those issued from the Evil account itself). While this is an expensive attack, it would result in slowdown. Coupled with colluding staked leaders this could force a partition and allow for network takeover by speeding up or removing the offending logic once the partition occurred. This could present a version of the “Whale” problem experience in other crypto currencies, this could result in enough movement in the short term of any related token, value store, or counter monitoring the traffic or token flow.

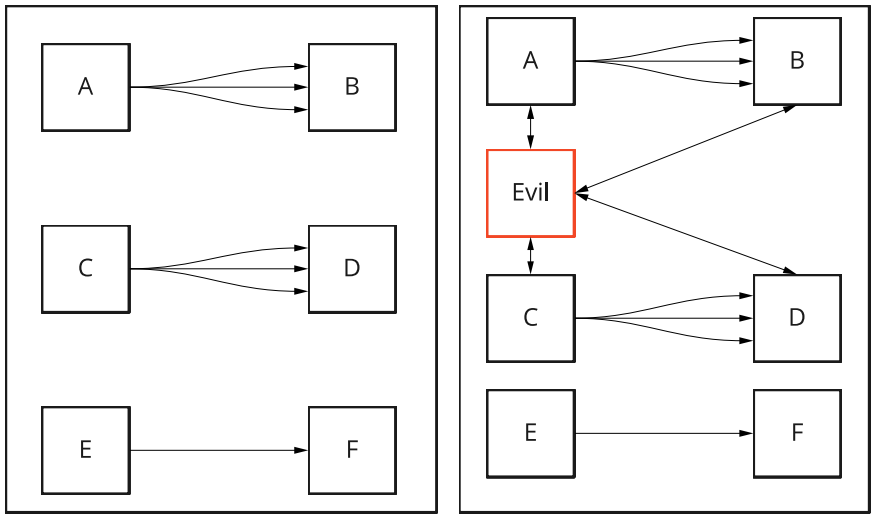


Figure 26: Sealevel "De-Optimizer"

6.6.2.5 All Accounts In Transaction

If a block explorer can be used to effectively mine for accounts, then it would be possible to submit transactions which forced operations (especially debit operations – even if the debit was 0) against all accounts. This would result in the resource limitation slowdowns and potential partitioning resulting from account overload discussed in other locations.

6.6.3 Innovation Dimension Ranking

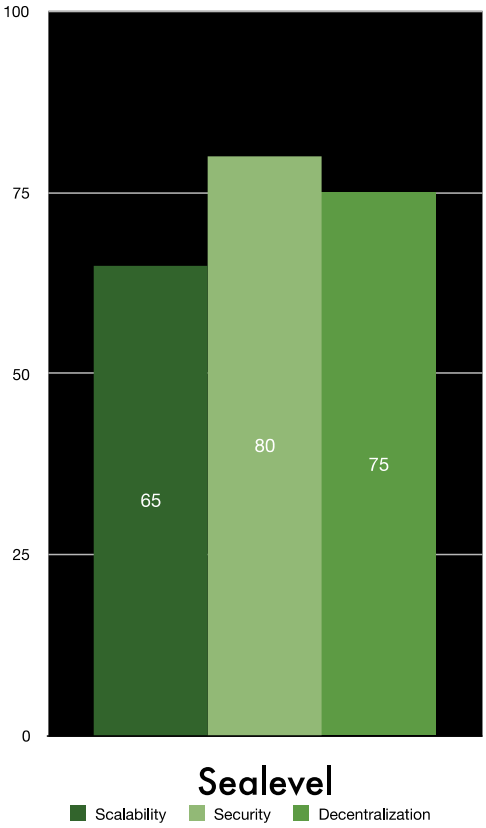


Figure 27: IDR Sealevel

For scalability, Sealevel needs to ensure that security best practices such as connection limits, authentication, verification, authorization controls, and other wireline and blockchain protocols are in place and tested in a heterogenous, scaled deployment. The behaviors or these contracts in those environments needs to be well understood.

The system maintains a good security posture thanks to its reliance on well-understood components in Berkeley Packet Filter and LLVM. This is a strong background and makes the innovation well-placed moving forward.

The aim is to bring Sealevel to a decentralized play, but like the scalability dimension, this needs to be proven and understood at scale in a heterogenous, permission-less realm. We suspect as well that the “lowest common denominator” problem (2.2) may be an issue for optimizations here at scale.

6.7 Innovation: Pipelining

6.7.1 General Description

[Solana] Pipelining is [a discrete, coordinated] process when there's a stream of input data that needs to be processed by a sequence of steps and there's different hardware responsible for each. To maximize efficiency, [Solana Pipelining] creates a pipeline of stages. … Given infinite loads, the pipeline will consistently complete a [work]load at the rate of the slowest stage in the pipeline. [Solana styles this four-stage transaction processor as the TPU, Solana's Transaction Processing Unit.]

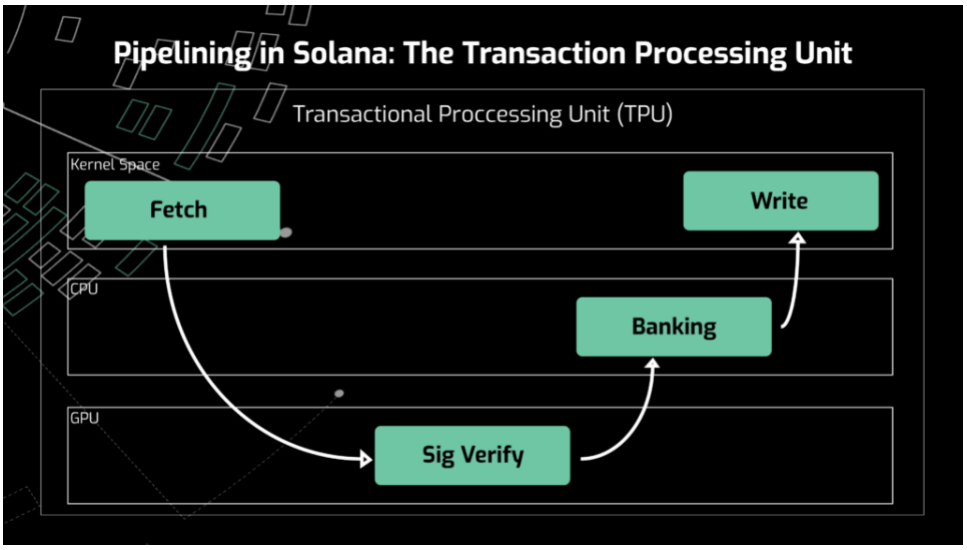


Figure 28: Pipelining and Transaction Processing Unit. Source: Solana

[Solana Pipelining[18]] progresses through Data Fetching at the kernel level, Signature Verification at the GPU level, Banking at the CPU level, and Writing [in] kernel space. By the time the TPU starts to send blocks out to the validators, it's already fetched in the next set of packets, verified their signatures, and begun crediting tokens.

The Validator node simultaneously runs two pipelined processes, one used in leader mode called the TPU and one used in validator mode called the TVU. … . The TPU exists to create ledger entries whereas the TVU exists to validate them.

6.7.2 Security Claim Findings

While the concept of pipelining is not new, we have a couple of concerns related to barriers and decisions as to when to move between pipelines and around the atomicity of pipelining itself.

[18] Generally speaking, the discussion over Pipelining and TPU is confusing in its terminology. Solana should consider consistent branding here, especially with both terms (pipeline and TPU) having rich histories beyond blockchain and Solana already.

6.7.2.1 Shared Components

If there are any components in the pipeline which are shared between transaction flows, then the speed at which other actions in the pipeline can be executed can create a resource starvation issue. For example, if banking is slower than verification, then verifying accounts with a combination of large or many transactions can result in a marked slowdown or even denial-of-service at the banking action. This is a well-understood “third party” risk scenario where an externalized resource unavailability causes the entire system to deadlock waiting for the external transaction. Any implementation of this waiting on an external resource should be avoided at implementation time. Where such waiting and locking is unavoidable, the behavior should be documented and well-understood.

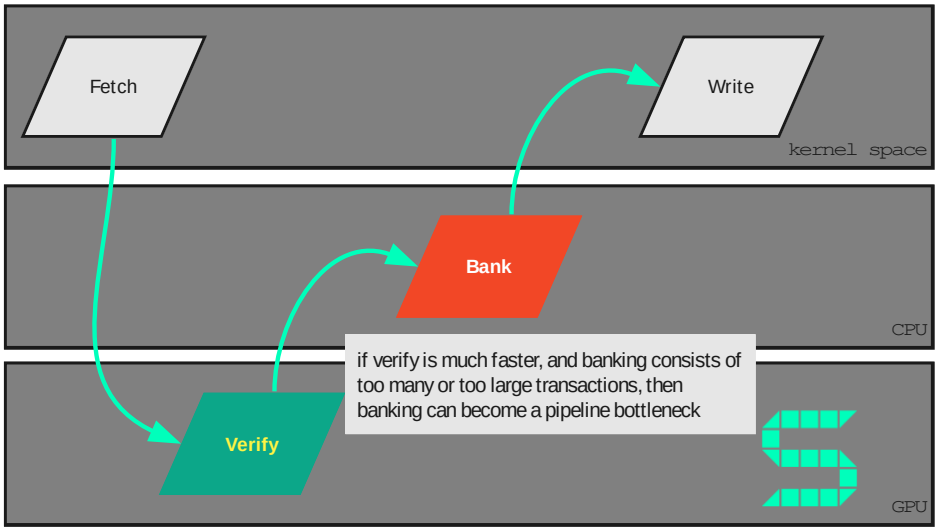


Figure 29: Pipeline Shared Component Bottlenecks

6.7.2.2 Parallel Pipelines

In the case where multiple transactional paths are at play, there exists at least one shared resource (a mutex in the blocktree held by the write stage). Any shared resources could result in issues with transaction atomicity, delays, ordering, or overloading. Additionally, in cases where parallel operations are handled by a lower-level driver (the VFS/disk IO layers for the Write Action for example), then how are failures and ordering misalignments being handled in parallel operations. Considerations related to network attached storage or other remote resources should also be considered here, especially when deployed on public cloud environments.

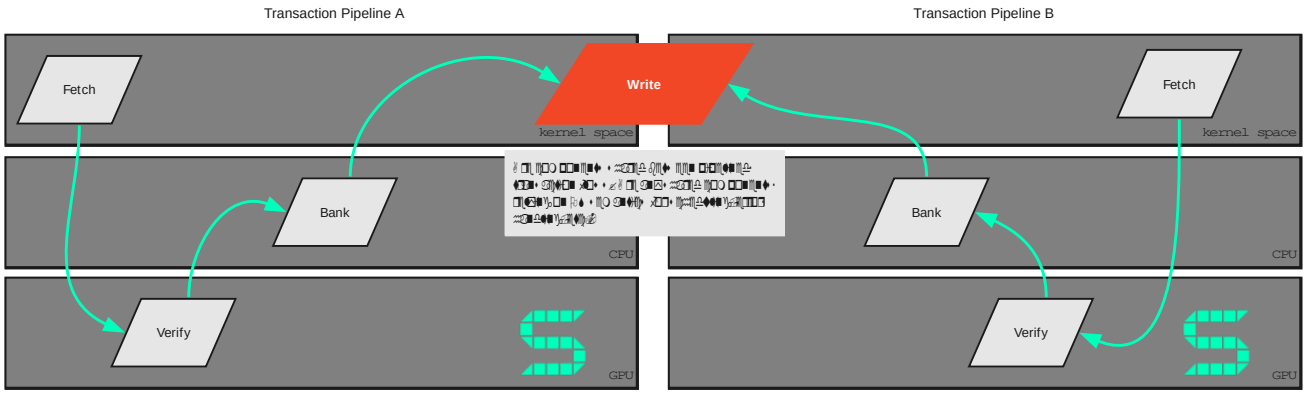


Figure 30: Parallel Pipeline Shared Actions

6.7.3 Innovation Dimension Ranking

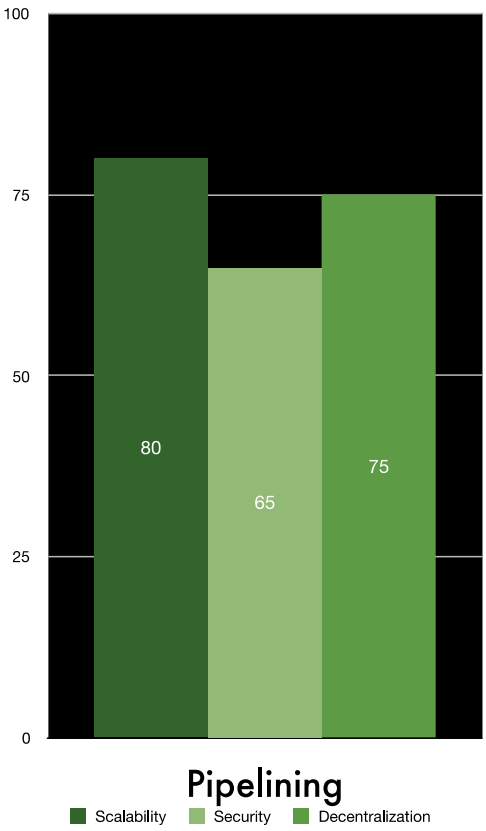


Figure 31: IDR Pipelining

Scalability:

Pipelining is a good optimization for a single node, with the caveat that using it for the Solana ecosystem requires having a node which is capable of leveraging it. This means having a GPU, a certain amount of memory, enough cores, etc. A heterogenous environment capable

of leveraging the optimization would require potential ports to other CPU architectures like ARM, supporting additional GPU types or alternate accelerator technologies like DSPs, FPGAs, etc.

Security:

Pipelining is well designed, but its reliance on shared components means that some aspects of security are outside the auspices of the improvement itself and handled by the resource orchestrator. For example, writes are coordinated by the operating system kernel.

Decentralization:

Pipelining does not aim to improve decentralization. And due to the "lowest common denominator" problem (2.2), ensuring that it does requires lowering this barrier to entry. Solana will need to work through how they want to position the improvement.

6.8 Innovation: Cloudbreak

6.8.1 General Description

[Solana first leverages] memory-mapped files[19]. A memory-mapped file is a file whose bytes are mapped into the virtual address space of a process. … [Solana secondly employs opportunistic parallelism to overcome sequential operation limitations.] [Cloudbreak exploits] this behavior [to] break up the accounts data structure roughly as follows:

1. *The index of accounts and forks is stored in RAM.*
2. *Accounts are stored in memory-mapped files up to 4MB in size.*
3. *Each memory map only stores accounts from a single proposed fork.*
4. *Maps are randomly distributed across as many SSDs as are available.*
5. *Copy-on-write semantics are used.*
6. *Writes are appended to a random memory map for the same fork.*
7. *The index is updated after each write is completed.*

… Cloudbreak also performs a form of garbage collection. As forks become finalized beyond rollback and accounts are updated, old invalid accounts are garbage collected, and memory is relinquished.

[19] Memory-mapped files exist in a variety of patterns by operating system. There is some general discussion at https://en.wikipedia.org/wiki/Memory-mapped_file. The particular implementation for Solana Cloudbreak leverages replicated blocks to construct persistence-backed files which are mapped into process virtual memory on initial load.

6.8.2 Security Claim Findings

6.8.2.1 Single Account Transactional Overload ("The iTunes Problem")

In cases where there are many transactions, especially many large transactions, against a single account, it is possible to overrun the capability of Cloudbreak since it does not allow sharding by account. We represent the problem in Figure 32. In these cases, the system would effectively take an account offline or cause large processing delays. If these exceed the block hash timeout, then those transactions would fail for the client. In use cases where a large account holder is the "largest customer" and does the vast majority of the transactions it could be possible for this account holder to experience a bottleneck in this scenario.

We recommend a mitigation strategy here much like that presented in the figure below where sub-accounts are leveraged to provide only a single credit transaction to the master at the end of the block. There would need to be investigation on whether or not a single credit could be compelled in a single block however, as the resulting transaction could also overrun and be forwarded without performing the credit. Either way, this situation should be investigated and better understood.

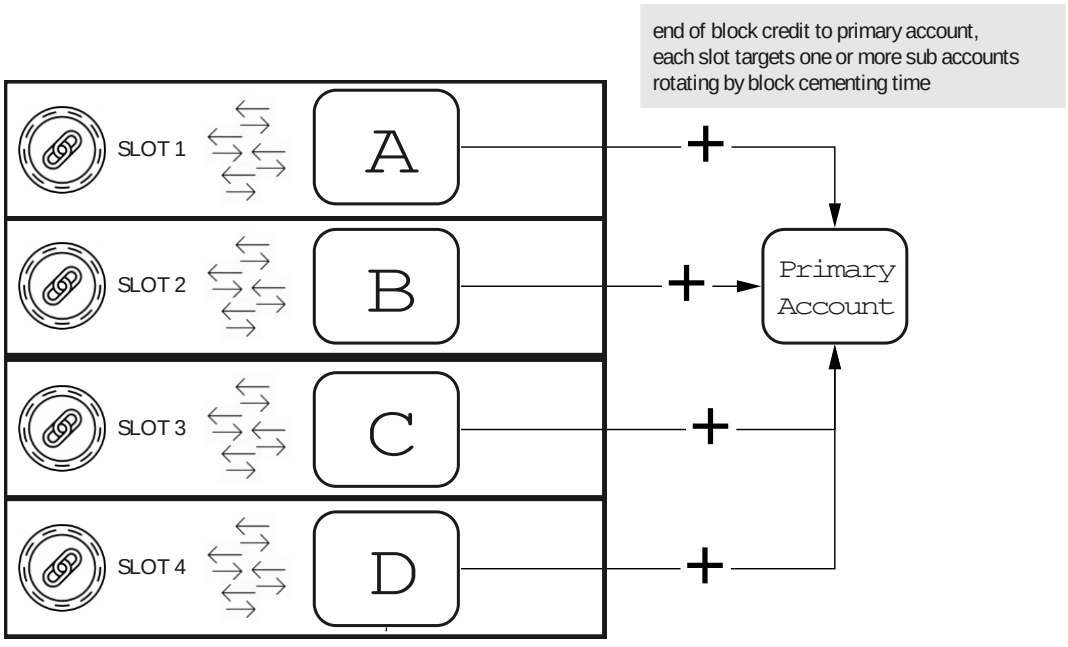


Figure 32: Single Account Transactional Overload

6.8.2.2 Replication Reconstruction

This system relies on the Turbine distribution to reassemble blocks when transactions are lost. This is true both for reconstruction and redistribution if there is a change to configuration or system layout. Need to understand the real implications of this – especially if compounded

with the iTunes problem where we have several accounts (or thousands of accounts) which are growing at rates which can overrun IO resources and thereby cause exhaustive transactions. The underlying filesystem, cloud infrastructure, or systems fragmentation could exacerbate this situation.

6.8.2.3 Intentional Garbage Creation ("Million Voices Problem")

Due to the ways in which garbage collection reaps, it could be possible to intentionally create millions of accounts in an effort to retire active accounts from the Cloudbreak platform and force disk churn to retrieve them. This would in turn slow down block confirmations and transaction processing on leaders and could lead to a cascade failure scenario. Account creation should be a rate-limited activity or be controlled by an appropriate business process. While this is unlikely in a permission-less system, it is crucial that the scenarios and behaviors for this activity be understood and mitigated as much as possible in such a deployments.

6.8.3 Innovation Dimension Ranking

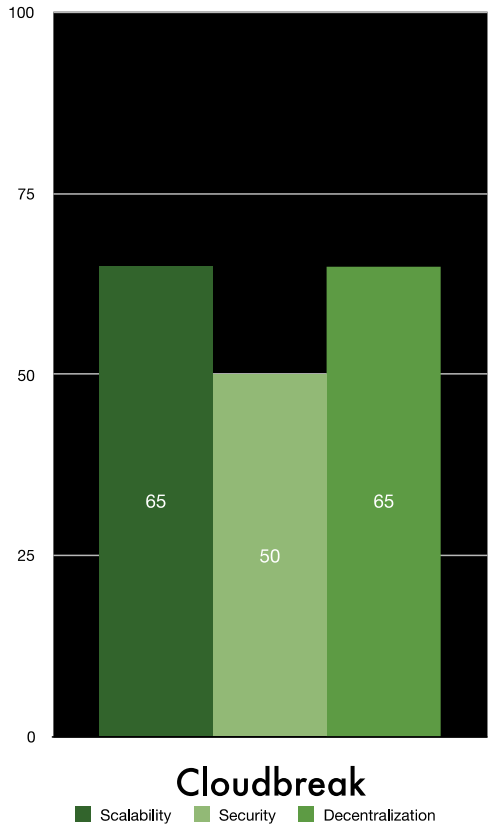


Figure 33: IDR Cloudbreak

Cloudbreak is primarily a data structure innovation for a single node. This works well for scalability but ties scaling to disk resources and so needs additional investigation, especially for a heterogenous system. Security is not a primary aim, and there are improvements which are required to rank this higher. See the findings in this section. While Cloudbreak does enhance account management, it is heavily dependent on resource constraints from other

components and so its impact on the decentralization of the entire system is questioned to some extent.

6.9 Innovation: Archivers

6.9.1 General Description

[Solana] Proof of History technology can be leveraged to [allow] a fast-to-verify implementation of Proof of Replication and enabling a bit torrent-esque distribution of the ledger across millions of Archiver nodes around the world. Archivers are not consensus participants, and have very low hardware requirements.

At a high level, the Solana Archiver network functions as follows: Archivers must signal to the network that they have X bytes of space available for storing data. On some frequency, the network divides the ledger history into pieces to target some replication rate (currently [anticipating] a target rate around 100x) and fault tolerance … based on the number of Archiver identities and total available storage of Archivers. Once Archiver:data assignments are made, each Archiver downloads her respective data from consensus validators. On some frequency, Archivers will be challenged to prove they're storing data, at which point they must complete a [Proof of Replication] (PoRep).

The basic idea of Proof of Replication is to encrypt a dataset with a public symmetric key using CBC encryption, and then hash the encrypted dataset[20]. [Solana randomly samples] the encrypted blocks at a faster pace than the speed of encryption, and [records] the hash of those samples into the PoH ledger. Thus the blocks stay in the exact same order for every PoRep and verification can stream the data and verify all the proofs in a single batch.

To begin producing PoReps of the ledger, the Archiver client does the following:

1. *Clients sign a PoH hash at a regular period*
2. *Signature is used as the source of randomness to pick a specific slice of the ledger*
3. *Signature is used to create a symmetric CBC key and the client encodes the slice of the ledger with the key.*

Since each client sligns the same PoH hash, the signatures are randomly distributed between all the clients. Clients then continuously sample the encrypted sample:

1. *Clients sign a PoH hash at a regular period.*
2. *Signature is used as the source of randomness to sample 1 byte per 1MB of the slice.*
3. *Samples are hashed with SHA256.*

[20] The original Solana blog post references the FileCoin discussion of replication at https://filecoin.io/proof-of-replication.pdf. We do not endorse or comment on the feasibility or efficacy of either implementation here, but focus solely on security aspects of the deployed system and developed code.

All the clients are forced to use the same PoH hash value as the signature. Since the signature tied to PoH, the resulting hash of samples is unique to that point in time and to that specific replication.

Validators in turn check the clients' proofs:

1. *Validator declares how many PoReps it can verify, based on number of GPU cores*
2. *Periodically validators will sign a PoH hash.*
3. *The signature is used to select a slice of the ledger to verify, and a mask to select which samples to verify up to the capacity of the validator.*
4. *Validator uploads the proofs that failed verification.*

A client can challenge a Validator for a failed proof by fishing for lazy validators. To prevent grinding attacks, clients must use the same keypair identity continuously. To prevent spam, all the messages in the protocol incur transaction fees.

6.9.2 Security Claim Findings

6.9.2.1 General Dependencies

The Archiver inherits all of the findings associated with both TowerBFT and Turbine since it relies on both technologies for its current implementation.

Additionally, while all of the innovations inherit the general findings, it is important to point out that resource-related issues are especially problematic for the Archivers since they generally work to help alleviate resource issues in other parts of the model.

6.9.2.2 "Replacing The Past" ("Doctor Who Attack")

One concern that was raised during our conversations is around the viability of leveraging the known hash chain to replace, obfuscate, or alter data or transactions in blocks between hashes. Since the archivers hold data which is already cemented in the block, but the hash chain is known, it could be possible for a malicious Archiver to replace the data in a block, but still report the known hash chain on relaying data back to a client. Without an active validator in that path, there would not be any way to prove the data was actually the original.

There was debate about the feasibility of this attack, but there was enough thought around the possibility that we document it here and represent it in the figure below.

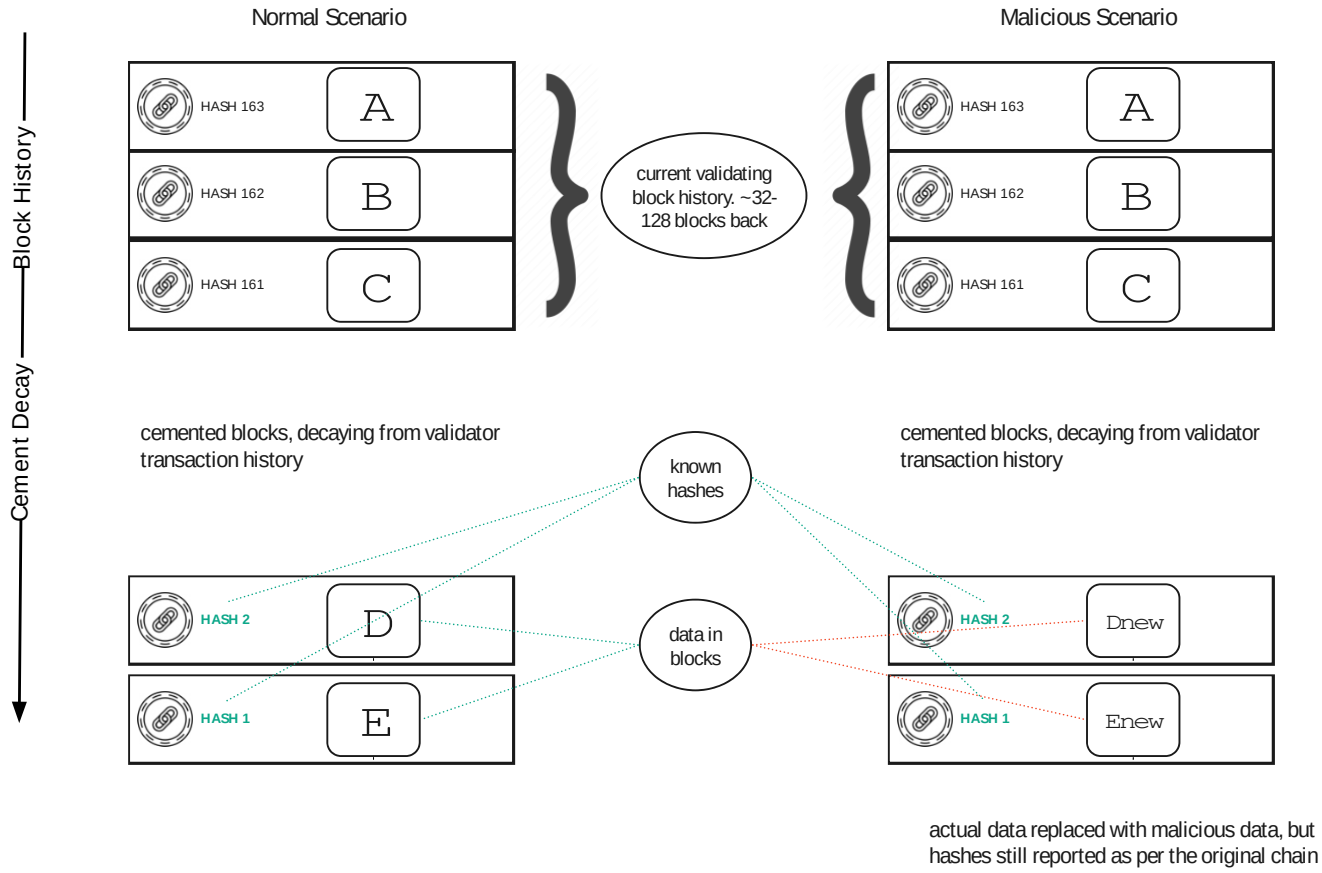


Figure 34: Replication, "Dr Who Problem"

6.9.2.3 CBC Chaining

For block replication, the system relies on the CBC elliptic curve cryptography called out in the general section above (5.6.2). See that discussion for context.

6.9.3 Innovation Dimension Ranking

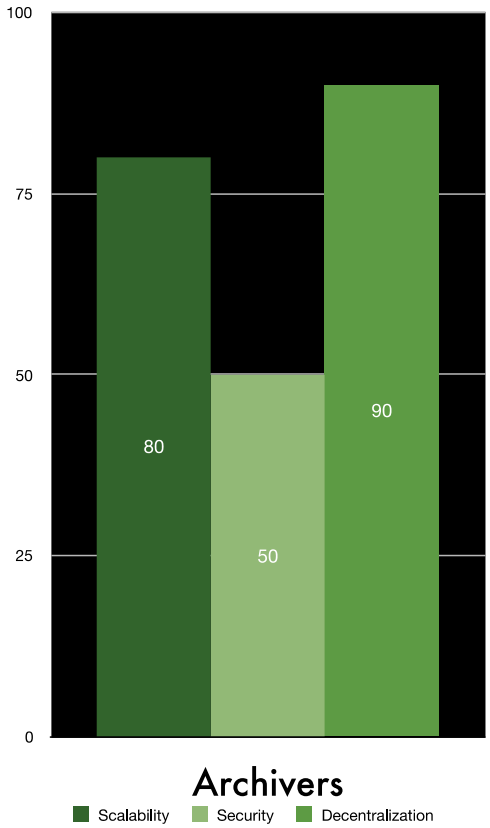


Figure 35: IDR Archivers

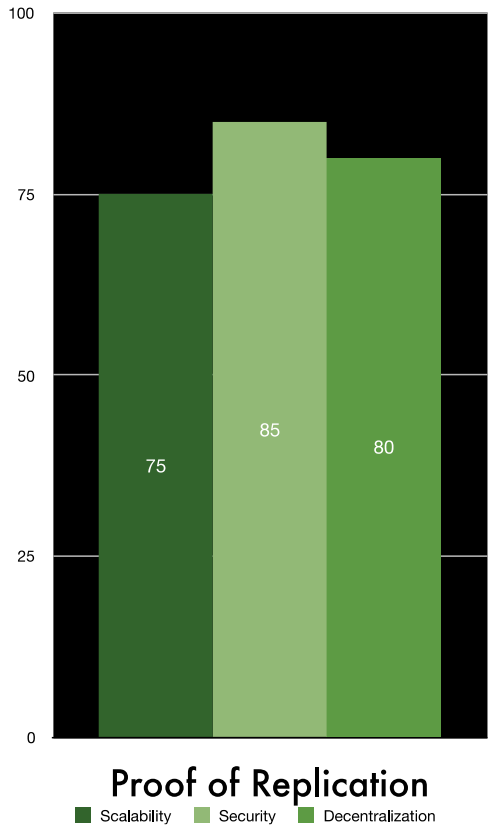


Figure 36: IDR Proof of Replication

Proof of Replication and Archivers essentially combine the attributes of Proof of History and Turbine and so inherit the comments from both. While security is not a key focus, it is important to note that the system essentially relies on external components to provide authority, verification, and validation and so will need to be understood in light of the implementation(s) of the system. This is a good decentralization target, but we still need to see it in a permission-less, heterogenous deployment at scale. It should be noted that one embodiment of the system could leverage centralized Archivers in the midst of an otherwise decentralized chain as both a security and performance enhancement.

6.10 Innovation Summary

Through the thorough analysis of the white papers, current implemented homogenous and (where available, heterogenous) systems and conversations, the analysis shows that there are limited security scenarios which would result in money loss from the system or theft of funds from accounts without a full takeover of the entire system such as node-exhaustion or collusion.

From a security perspective, this system has limited high-risk findings pending review of the final code of the system and of course relies on appropriate controls being implemented at

built time on the physical systems, cloud nodes, and other corresponding human and systems environments.

7 Appendix A: About Kudelski Security

Kudelski Security is an innovative, independent Swiss provider of tailored cyber and media security solutions to enterprises and public sector institutions. Our team of security experts delivers end-to-end consulting, technology, managed services, and threat intelligence to help organizations build and run successful security programs. Our global reach and cyber solutions focus is reinforced by key international partnerships.

Kudelski Security is a division of Kudelski Group. For more information, please visit https://www.kudelskisecurity.com.

Kudelski Security

route de Genève, 22-24

1033 Cheseaux-sur-Lausanne

Switzerland

Kudelski Security

5090 North 40th Street

Suite 450

Phoenix, Arizona 85018

8 Appendix B: Blockchain Trilemma

The Solana team has asked for the position of their applications and developed innovations with regards to Vitalik Buterin's famous Blockchain Trilemma. Towards that end, we will do our best to try and provide justification for where we see the innovations falling on both maps, with a short discussion of those reasons in each section. Figure 37 shows the traditional Blockchain Trilemma diagram as envisioned by Buterin[21] and followed through lengthy discussions elsewhere.

We will structure the document to include a miniature version of the Trilemma scale in each section to allow for directed discussion by the Solana team with whatever audiences they deem necessary.

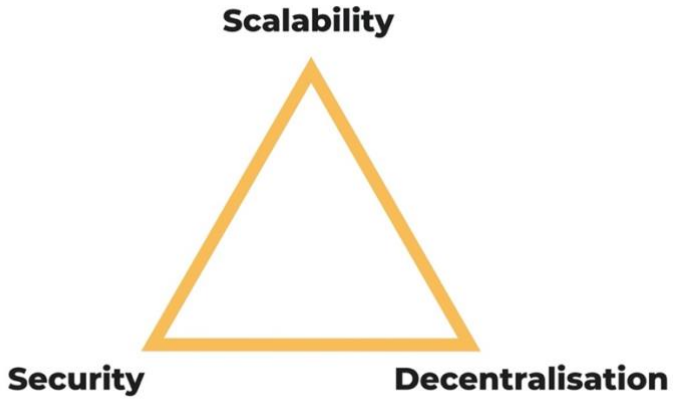


Figure 37: The traditional Blockchain Trilemma triangle. Source: medium.com

8.1.1 Blockchain Trilemma Placement Guidelines

Figure 38 outlines the overall placement of all of the Solana innovations ranked against the overall blockchain trilemma. It is important to note a few considerations in rankings:

- All rankings are on a 100-point (percentile) scale, with 50 being exactly between logical endpoints. Since this is a trilemma diagram, it is not possible for any single component to score a perfect 100 or 0 on any dimension.

[21] There are many places to get a discussion of the trilemma. The original can be found at https://github.com/ethereum/wiki/wiki/Sharding-FAQ#this-sounds-like-theres-some-kind-of-scalability-trilemma-at-play-what-is-this-trilemma-and-can-we-break-through-it and there are good general discussions to be found at https://medium.com/coinmonks/the-real-blockchain-trilemma-58824b52fe1d and https://cryptoticker.io/en/blockchain-trilemma-explained/ and https://www.coinreview.com/blockchain-trilemma/ . We are not endorsing any particular view in this document save to explain our rationale for security placement, but wanted to ensure that the discussion for the reader was relatively complete. We do not endorse any particular opinion over another for the purposes of this evaluation.

- All of the rankings along the blockchain trilemma are generally complete platforms and not individual innovations within the context of a larger platform. Since we are looking at the Solana innovations individually some components of the trilemma don't make sense for some of the innovations. For example, some innovations are meant to enhance performance of a local leader node but are not intended to affect decentralization at all. In cases like this, we rank those innovations a "50" to locate them directly in the middle of a dimension. Consider this to be a "neither here nor there" ranking.
- *Scalability* generally refers to the capability of a blockchain to increase the number of transactions per second against the chain. It does not have a metric regarding the availability or resiliency of the chain itself. However, since scalability of the platform is directly tied to and dependent on the behavior of physical resources, our analysis includes availability and resiliency of the component in the scalability score. This could also be a facet that readers of this report are interested in.
- Dimensions sway between two logical concepts. These are as follows:
    - Scalability swings between infinitely scalable in terms of transactions per second but with no reliability (0) and completely reliable but no transactions per second against the chain (100). Scalability is measured on the left side of the triangle.
    - Security swings between no security and completely open (0) to fully locked down and closed (100). Security is measured on the bottom of the triangle.
    - Decentralization swings between completely decentralized with no centralized governance (100) and completely permissioned with no distributed autonomy (0) and is measured on the right side of the triangle.

8.1.2 Overall Placement of Innovations

Figure 38 shows a version of the blockchain trilemma with the associated Solana innovations called out as particular nodes on the ternary plot. We developed a model which assessed each component according to the scoring methodology outlined in 8.1.1. Each individual innovation score will be discussed in the section for that innovation.

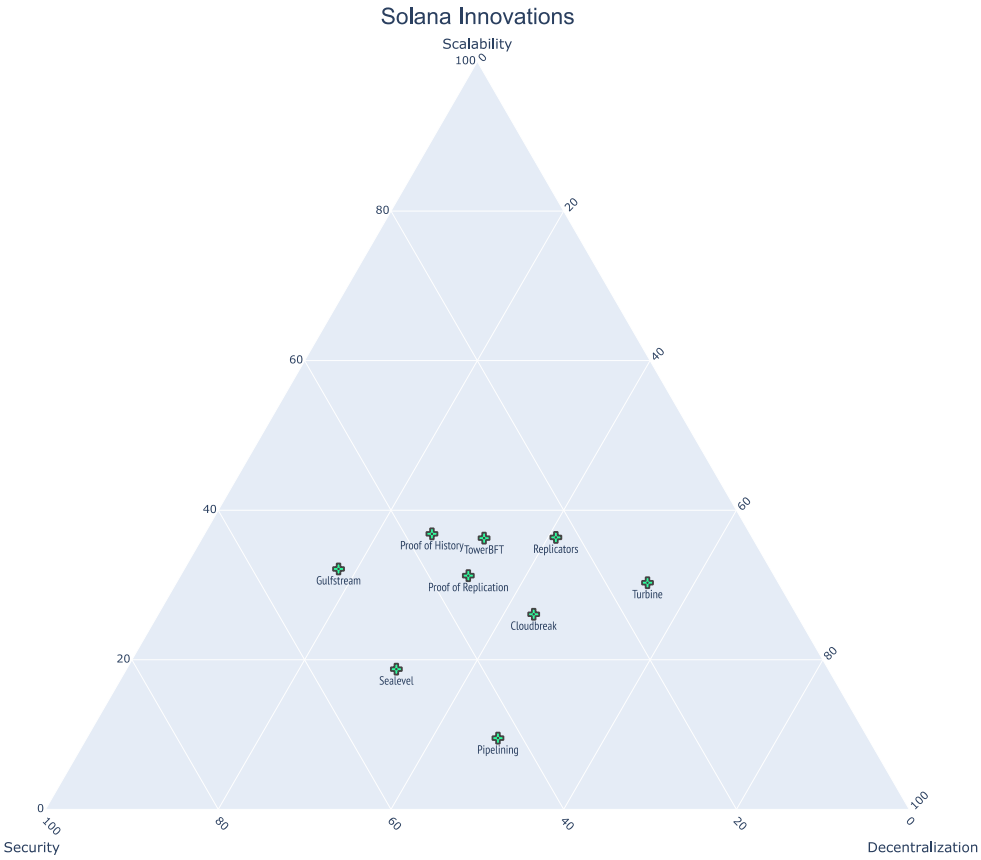


Figure 38: Solana Blockchain Trilemma Placements

8.1.3 Proof of History Blockchain Trilemma Position

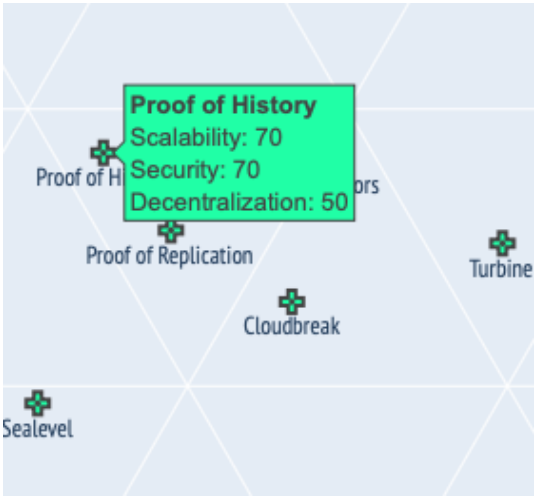


Figure 39: Blockchain Trilemma, Proof of History

Proof of History aims to provide a new, enhanced mechanism for consensus and externalized timing which does not require direct wall-clock synchronization. It is a technology aimed

directly at providing for a new means to create decentralization while benefiting from the speed of centralized and distributed agreement. Towards that end we rank the system as

Scalability: 70

As designed, the PoH idea represents a good approach to increasing the number of transactions per second to the chain since the hash rate is configurable and can be tuned to the performance of the nodes in the platform. This can be made available by agreeing on the range of available CPUs and GPUs ahead of time and horizontally scaling.

Security: 70

Leveraging SHA-256 creates a wide hash space and the addition of a per-block salt which changes based on input, allows for a high degree of security. While there are attacks which could compromise single blocks, the ability to force a fork in the chain or to corrupt the entire PoH tick stream is difficult.

Decentralization: 50

While the aim of the PoH technology is to enhance decentralization, there is a great deal of reliance on homogeneity in the deployed infrastructure and testing with a variety of latency boundaries and intermittent behavior is not complete. The structure will perform well in permissioned platforms but is not yet proven in a permission-less structure, so we rank the current model in the middle of the two.

8.1.4 TowerBFT Blockchain Trilemma Position

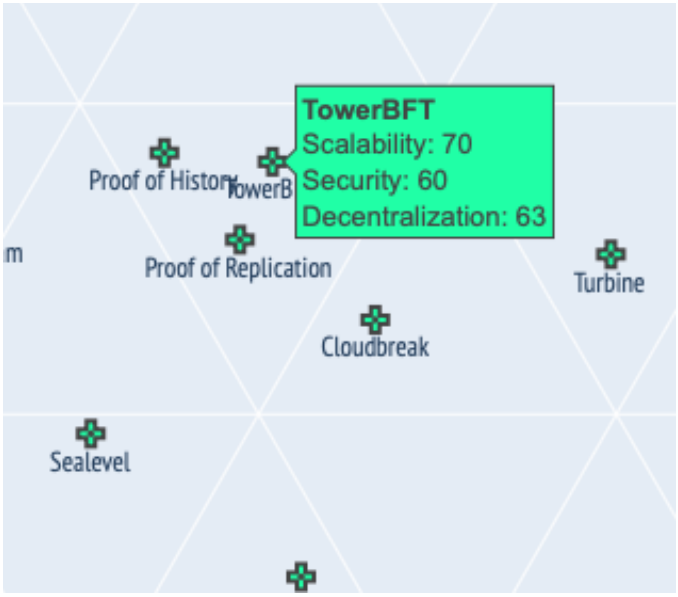


Figure 40: Blockchain Trilemma, TowerBFT

TowerBFT is the consensus algorithm which leverages PoH. It is designed to provide a decentralized method for quick convergence in practical terms. For innovations on the blockchain trilemma we score it as:

Scalability: 70

This inherits essentially the same attributes as PoH. Likewise, Practical BFT has proven itself to represent reasonable availability with increased transactional workload and performs admirably when handling network partitions.

Security: 60

There are many scenarios in TowerBFT where attacking leaders, especially with colluding tick sources for PoH result in forks, denial of service, or other problems. Accordingly, while this is a good approach to Practical BFT, we feel that more of the solutions and mitigations should be in place before scoring this higher.

Decentralization: 63

Leveraging the proven capabilities of PBFT to move towards a distributed and decentralized position, the true permission-less capabilities are yet to be proven in the wider blockchain world. As a result, we score this higher than straight PoH, but we need to see a more proven behavior in the real world and mitigation and solutions for some of the issues raised — especially with regard to network partitions and repeated partitions — before we can say that it will function in a permission-less and fully decentralized deployment.

8.1.5 Turbine Blockchain Trilemma Position

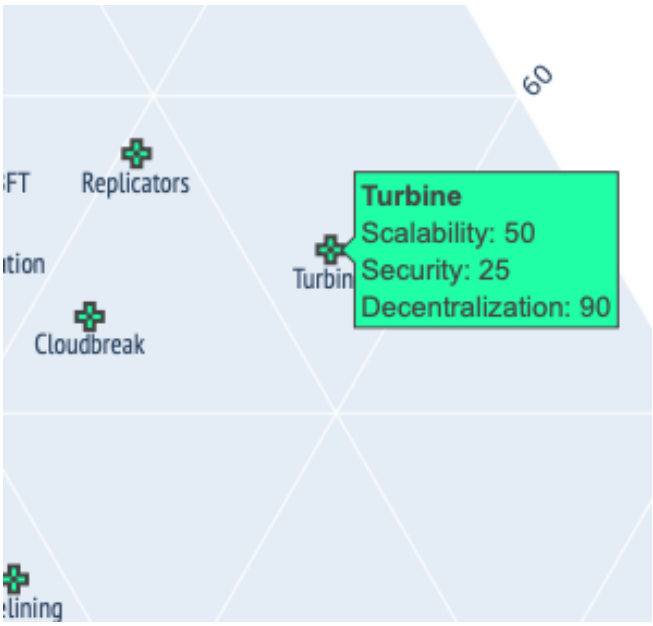


Figure 41: Blockchain Trilemma, Turbine

We score Turbine according to the following considerations.

Scalability: 50

Turbine finds itself in the middle of the road for scalability. Certainly, it aims to provide high transactional throughput, though it is not specifically designed to enhance transactions per

second, but instead to speed up block confirmation and verification. It does this admirably. Likewise, it does handle failure recovery as long as the distribution network is functioning properly. There are several scenarios where blocked paths can overrun the network capabilities or prevent nodes from receiving entire blocks, however, which then affect the overall availability and stability of the platform.

Security: 25

Turbine essentially trusts everyone in the participating network list. This is an aim towards speed and recoverability but not one which aims for security at the core. There is an effort to obfuscate and make it more difficult to attack the block sources by shuffling the participant list for each shred, but this is not a directed effort at overall security for the environment. As said though, this is not a core goal for the innovation.

Decentralization: 90

Turbine's core aim is to provide fast, decentralized distribution of blocks and it scores well on that aim.

8.1.6 Gulfstream Blockchain Trilemma Position

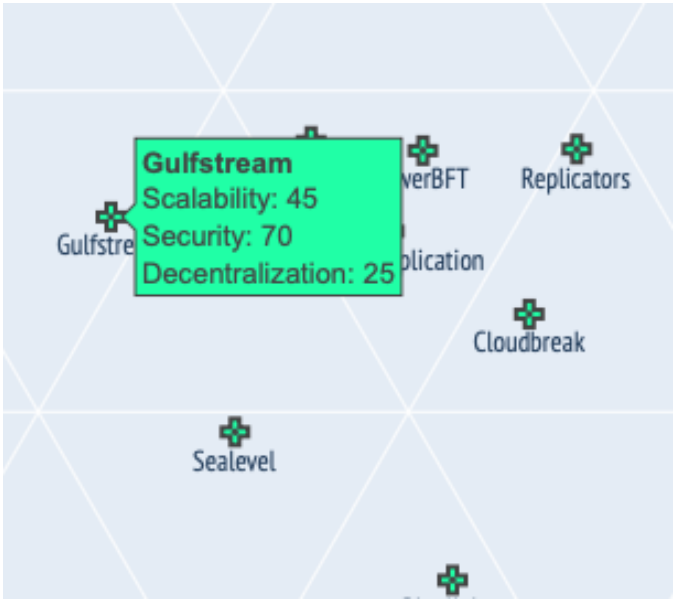


Figure 42: Blockchain Trilemma, Gulfstream

Gulfstream is an intentional move to speed up unconfirmed transactions against the blockchain and as such aims to improve transaction throughput. However, the reliance on client behavior to handle failure scenarios and single-threaded performance of a leader to add the actual transactions means that its impact could be limited.

Scalability: 45

Since the Gulfstream technology is an unconfirmed forwarding technology, it does not impact the capability of the leader to actually confirm transactions. Indeed, we suspect that there may

be situations where the mechanism can actually overload leaders. Likewise, the method is specific to individual validators and has no distributed availability component, making it ultimately fall in the “middle of the road” on the scale.

Security: 70

Since Gulfstream is specific to a leader and does not communicate to other nodes at all, it is difficult to attack externally. Aside from the resource-constraint based problems mentioned above, the system is limited to a single node.

Decentralization: 25

Gulfstream is node-specific and does not aim to improve the overall permission-less state of the system.

8.1.7 Sealevel Blockchain Trilemma Position

Figure 43: Blockchain Trilemma, Sealevel

The smart contracts implementation leverages the well understood BPF interface, but in a new rust implementation. Accordingly, some testing for both transactional scaling and reliability remain to be done over time. Additionally, the implementation does not aim to improve on the decentralization of the product beyond providing an egalitarian interface across platform implementations. We score Sealevel as:

Scalability: 30

Security: 80

The BPF interface is well understood and trusted in a variety of security applications including firewalls, packet analyzers, routers, bridges, and the like.

Decentralization: 50

8.1.8 Pipelining Blockchain Trilemma Position

Figure 44: Blockchain Trilemma, Pipelining

On the trilemma scale, we rank Solana Pipelining as:

Scalability: 55

The goal of the innovation is to enhance performance within a node. It does not speak at all to reliability. In fact, in some cases where there are limited node resources, or single resources in the case of CPU or GPU, the pipeline functionality may realize lower reliability overall. Towards this end, while the scalability does not directly improve the throughput for the overall blockchain deployment, in platforms where the node performance governed such that Pipelining optimizations are possible in all cases, it will improve throughput greatly.

Security: 65

Pipelining aims to improve throughput within a node and does not focus on security. There are some guards in place – mostly through the rust compiler semantics – which do focus on providing thread safety, but generally speaking security is not an aim here. That being said, the method and implementation are well understood by those familiar in the arts and do not represent many places for exploitation. There are some vulnerabilities where resources might be shared or governed outside the boundaries of the Pipelining configuration (e.g. the kernel might be managing the write operations for the entire system), but those are outside the scope of this analysis and also well-understood.

Decentralization: 50

Since node-specific performance enhancements are useful, they do not speak to the behavior of a distributed platform. Since staking is not determined by performance, this lands in the middle of the scale as it does not enhance or detract along this axis.

It is important to note here that Pipelining does not improve the overall decentralization play. Rather it would tend towards centralization if left as an ungoverned, unenforced optimization since only the most expensive, most accelerated platforms would benefit. For example, an Intel NUC would not benefit from Pipelining as it could not include a GPU which can be leveraged and cannot increase memory enough to make optimizations there relevant. With governance of some kind (stake benefits, preferred leadership through performance, etc.) this would tend to improve the system overall. As it stands for this analysis however, we see it as middle-of-the-road and rank accordingly.

8.1.9 Cloudbreak Blockchain Trilemma Position

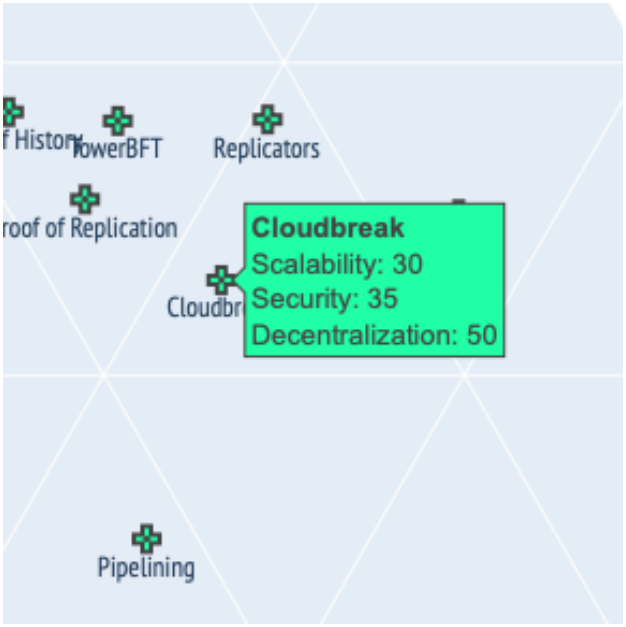


Figure 45: Blockchain Trilemma, Cloudbreak

Cloudbreak is an enhancement to the accounts management arm of the blockchain implementation. It is directly involved in the throughput and organization of transactions as they are inserted into blocks and subsequently verified.

Scalability: 30

There is no doubt that cloudbreak can speed up transactions on the leader, but there are scenarios which warrant additional testing to ensure they cannot adversely impact the behavior.

Security: 35

This is not a stated goal for Cloudbreak, as it deals only with account maintenance.

Decentralization: 50

This is not a stated goal for Cloudbreak, and as a node-specific enhancement, does not speak to behavior on a parallel, distributed computing scale.

8.1.10 Archivers Blockchain Trilemma Position

Figure 46: Blockchain Trilemma, Archivers

We score Archivers as:

Scalability: 80

The system employs well-understood and established means for distribution and sharing with K-factors that can scale horizontally as well as vertically.

Security: 50

The system relies on external overlays and behaviors brought to the Archiver platform for security and so lies in the middle of this axis.

Decentralization: 90

The primary goal for Archivers is to enhance decentralization and to speed up blockchain loading times for new validators and potentially for “light clients”. It does this admirably.

Figure 47: Blockchain Trilemma, Proof of Replication

We call out PoRep as a distinct concept from Archivers as its stated aims are specifically to provide enhancements to the consensus and agreement methods from which it derives.

Scalability: 75

Inheriting both capabilities from PoH as well as Archiving, the reliability and throughput of the mechanism can be made to be very strong.

Security: 85

As with scalability, since overlays are employed for this, the mechanism can score well here as well. While there are concerns over the security of the actual protected assets, this is not relevant for the protocol discussion[22].

Decentralization: 80

The key goal for the system is to enhance verifiable trust in a permission-less system and it does accomplish that.

[22] We should mention here however, that there are many regulations, laws, and legal opinions coming to light around the world with regards to the content which is hosted in permission-less systems. This is out of scope for the document at hand, but we recommend that the Solana team pay special attention to the winds here, as they may well blow towards the responsibility of the platform maintainer. For example, there have been recent developments in some countries which hold the distribution engines responsible for illegal content such as child pornography. We recommend caution in operating systems which provide anonymous and encrypted file storage generally.

9 Appendix C: Solana Whitepaper

The Solana Whitepaper (2017) is available online, but is attached here to provide additional context and explanation to the concepts, designs, architecture, and implementations in this document.

Solana: A new architecture for a high performance blockchain v0.8.14

Anatoly Yakovenko
anatoly@solana.io

Abstract

This paper proposes a new blockchain architecture based on Proof of History (PoH) - a proof for verifying order and passage of time between events. PoH 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 in sub-second finality times. This paper also proposes two algorithms that leverage the time keeping properties of the PoH ledger - a PoS algorithm that can recover from partitions of any size and an efficient streaming Proof of Replication (PoRep). The combination of PoRep and PoH provides a defense against forgery of the ledger with respect to time (ordering) and storage. The protocol is analyzed on a 1 gbps network, and this paper shows that throughput up to 710k transactions per second is possible with today's hardware.

1 Introduction

Blockchain is an implementation of a fault tolerant replicated state machine. Current publicly available blockchains do not rely on time, or make a weak assumption about the participant's abilities to keep time [4, 5]. Each node in the network usually relies on their own local clock without knowledge of any other participants clocks in the network. The lack of a trusted source of time means that when a message timestamp is used to accept or reject a message, there is no guarantee that every other participant in the network will make the exact same choice. The PoH presented here is designed to create a ledger with verifiable passage of time, i.e. duration between events and message ordering. It is anticipated that every node in the network will be able to rely on the recorded passage of time in the ledger without trust.

2 Outline

The remainder of this article is organized as follows. Overall system design is described in Section 3. In depth description of Proof of History is described in Section 4. In depth description of the proposed Proof of Stake consensus algorithm is described in Section 5. In depth description of the proposed fast Proof of Replication is described in Section 6. System Architecture and performance limits are analyzed in Section 7. A high performance GPU friendly smart contracts engine is described in Section 7.5

3 Network Design

As shown in Figure 1, at any given time a system node is designated as Leader to generate a Proof of History sequence, providing the network global read consistency and a verifiable passage of time. The Leader sequences user messages and orders them such that they can be efficiently processed by other nodes in the system, maximizing throughput. It executes the transactions on the current state that is stored in RAM and publishes the transactions and a signature of the final state to the replications nodes called Verifiers. Verifiers execute the same transactions on their copies of the state, and publish their computed signatures of the state as confirmations. The published confirmations serve as votes for the consensus algorithm.

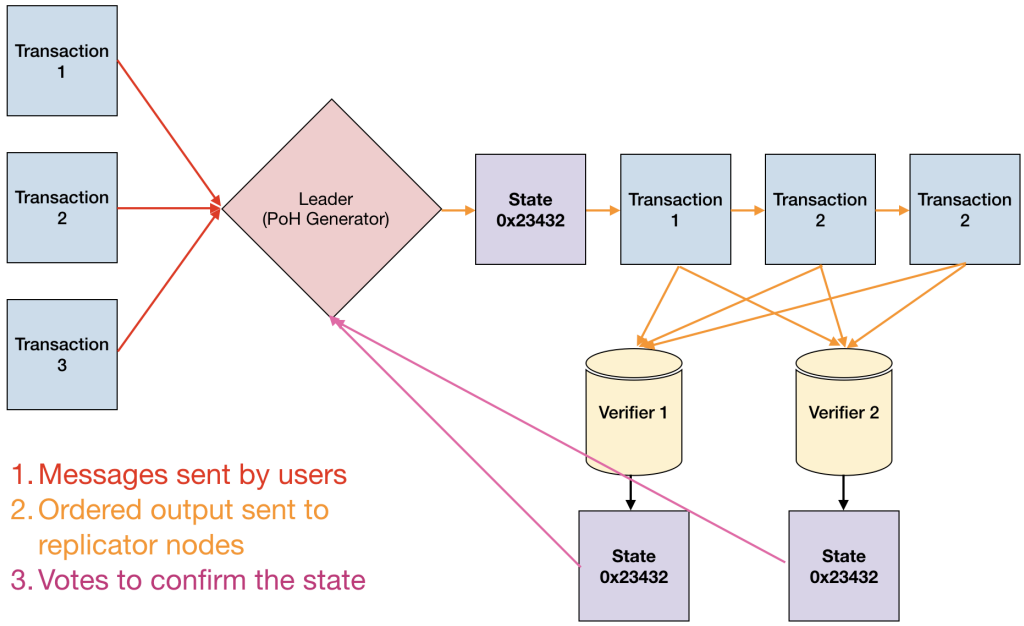


Figure 1: Transaction flow throughout the network.

In a non-partitioned state, at any given time, there is one Leader in the network. Each Verifier node has the same hardware capabilities as a Leader and can be elected as a Leader, this is done through PoS based elections. Elections for the proposed PoS algorithm are covered in depth in Section 5.6.

In terms of CAP theorem, Consistency is almost always picked over Availability in an event of a Partition. In case of a large partition, this paper proposes a mechanism to recover control of the network from a partition of any size. This is covered in depth in Section 5.12.

4 Proof of History

Proof of History is a sequence of computation that can provide a way to cryptographically verify passage of time between two events. It uses a cryptographically secure function written so that output cannot be predicted from the input, and must be completely executed to generate the output. The function is run in a sequence on a single core, its previous output as

the current input, periodically recording the current output, and how many times it's been called. The output can then be re-computed and verified by external computers in parallel by checking each sequence segment on a separate core.

Data can be timestamped into this sequence by appending the data (or a hash of some data) into the state of the function. The recording of the state, index and data as it was appended into the sequences provides a timestamp that can guarantee that the data was created sometime before the next hash was generated in the sequence. This design also supports horizontal scaling as multiple generators can synchronize amongst each other by mixing their state into each others' sequences. Horizontal scaling is discussed in depth in Section 4.4

4.1 Description

The system is designed to work as follows. With a cryptographic hash function, whose output cannot be predicted without running the function (e.g. `sha256`, `ripemd`, etc.), run the function from some random starting value and take its output and pass it as the input into the same function again. Record the number of times the function has been called and the output at each call. The starting random value chosen could be any string, like the headline of the New York times for the day.

For example:

PoH Sequence

Index	Operation	Output Hash
1	sha256("any random starting value")	`hash1`
2	sha256(`hash1`)	`hash2`
3	sha256(`hash2`)	`hash3`

Where `hashN` represents the actual hash output.

It is only necessary to publish a subset of the hashes and indices at an interval.

For example:

PoH Sequence

Index	Operation	Output Hash
1	sha256(”any random starting value”)	`hash1`
200	sha256(`hash199`)	`hash200`
300	sha256(`hash299`)	`hash300`

As long as the hash function chosen is collision resistant, this set of hashes can only be computed in sequence by a single computer thread. This follows from the fact that there is no way to predict what the hash value at index 300 is going to be without actually running the algorithm from the starting value 300 times. Thus we can thus infer from the data structure that real time has passed between index 0 and index 300.

In the example in Figure 2, hash `62f51643c1` was produced on count 510144806912 and hash `c43d862d88` was produced on count 510146904064. Following the previously discussed properties of the PoH algorithm, we can trust that real time passed between count 510144806912 and count 510146904064.

4.2 Timestamp for Events

This sequence of hashes can also be used to record that some piece of data was created before a particular hash index was generated. Using a ‘combine‘ function to combine the piece of data with the current hash at the current index. The data can simply be a cryptographically unique hash of arbitrary event data. The combine function can be a simple append of data, or any operation that is collision resistant. The next generated hash represents a timestamp of the data, because it could have only been generated after that specific piece of data was inserted.

For example:

PoH Sequence

Index	Operation	Output Hash
1	sha256(”any random starting value”)	`hash1`
200	sha256(`hash199`)	`hash200`
300	sha256(`hash299`)	`hash300`

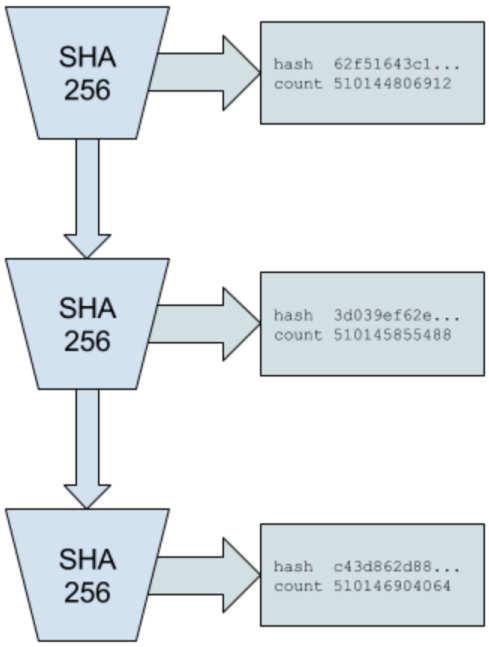


Figure 2: Proof of History sequence

Some external event occurs, like a photograph was taken, or any arbitrary digital data was created:

PoH Sequence With Data

Index	Operation	Output Hash
1	sha256("any random starting value")	`hash1`
200	sha256(`hash199`)	`hash200`
300	sha256(`hash299`)	`hash300`
336	sha256(append(`hash335`, photograph_sha256))	`hash336`

`Hash336` is computed from the appended binary data of `hash335` and the `sha256` of the photograph. The index and the `sha256` of the photograph are recorded as part of the sequence output. So anyone verifying this sequence can then recreate this change to the sequence. The verifying can still be done in parallel and it's discussed in Section 4.3

POH Sequence

Index	Operation	Output Hash
1	sha256(”any random starting value”)	`hash1`
200	sha256(hash199)	`hash200`
300	sha256(hash299)	`hash300`
336	sha256(append(hash335, photograph1_sha256))	`hash336`
400	sha256(hash399)	`hash400`
500	sha256(hash499)	`hash500`
600	sha256(append(hash599, photograph2_sha256))	`hash600`
700	sha256(hash699)	`hash700`

Table 1: PoH Sequence With 2 Events

Because the initial process is still sequential, we can then tell that things entered into the sequence must have occurred sometime before the future hashed value was computed.

In the sequence represented by Table 1, `photograph2` was created before `hash600`, and `photograph1` was created before `hash336`. Inserting the data into the sequence of hashes results in a change to all subsequent values in the sequence. As long as the hash function used is collision resistant, and the data was appended, it should be computationally impossible to pre-compute any future sequences based on prior knowledge of what data will be integrated into the sequence.

The data that is mixed into the sequence can be the raw data itself, or just a hash of the data with accompanying metadata.

In the example in Figure 3, input `cfd40df8…` was inserted into the Proof of History sequence. The count at which it was inserted is 510145855488 and the state at which it was inserted is `3d039eef3`. All the future generated hashes are modified by this change to the sequence, this change is indicated by the color change in the figure.

Every node observing this sequence can determine the order at which all events have been inserted and estimate the real time between the insertions.

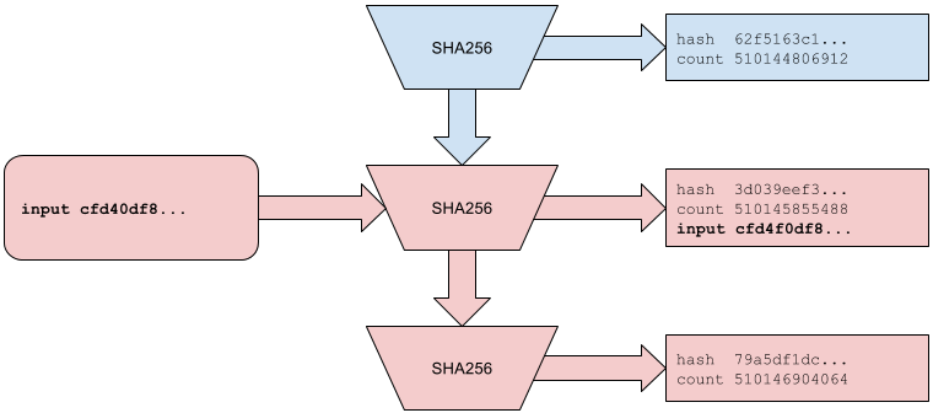


Figure 3: Inserting data into Proof of History

4.3 Verification

The sequence can be verified correct by a multicore computer in significantly less time than it took to generate it.

For example:

Core 1		
Index	Data	Output Hash
200	sha256(`hash199`)	`hash200`
300	sha256(`hash299`)	`hash300`
Core 2		
Index	Data	Output Hash
300	sha256(`hash299`)	`hash300`
400	sha256(`hash399`)	`hash400`

Given some number of cores, like a modern GPU with 4000 cores, the verifier can split up the sequence of hashes and their indexes into 4000 slices, and in parallel make sure that each slice is correct from the starting hash to the last hash in the slice. If the expected time to produce the sequence is going to be:

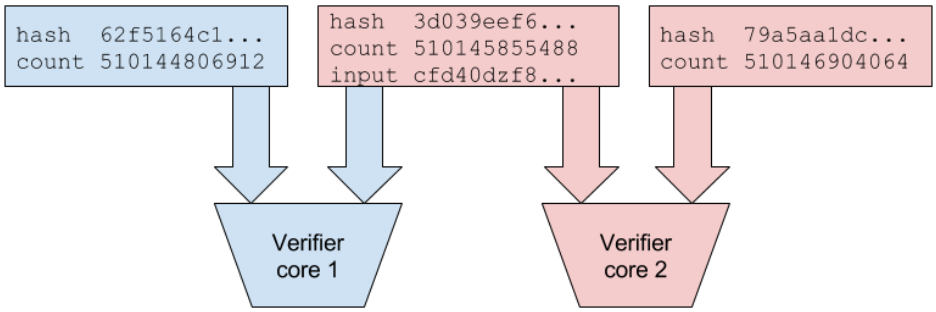


Figure 4: Verification using multiple cores

$$\frac{\text{Total number of hashes}}{\text{Hashes per second for 1 core}}$$

The expected time to verify that the sequence is correct is going to be:

$$\frac{\text{Total number of hashes}}{\text{(Hashes per second per core * Number of cores available to verify)}}$$

In the example in Figure 4, each core is able to verify each slice of the sequence in parallel. Since all input strings are recorded into the output, with the counter and state that they are appended to, the verifiers can replicate each slice in parallel. The red colored hashes indicate that the sequence was modified by a data insertion.

4.4 Horizontal Scaling

It's possible to synchronize multiple Proof of History generators by mixing the sequence state from each generator to each other generator, and thus achieve horizontal scaling of the Proof of History generator. This scaling is done without sharding. The output of both generators is necessary to reconstruct the full order of events in the system.

PoH Generator A		
Index	Hash	Data
1	`hash1a`	
2	`hash2a`	`hash1b`
3	`hash3a`	
4	`hash4a`	

PoH Generator B		
Index	Hash	Data
1	`hash1b`	
2	`hash2b`	`hash1a`
3	`hash3b`	
4	`hash4b`	

Given generators A and B, A receives a data packet from B (hash1b), which contains the last state from Generator B, and the last state generator B observed from Generator A. The next state hash in Generator A then depends on the state from Generator B, so we can derive that hash1b happened sometime before hash3a. This property can be transitive, so if three generators are synchronized through a single common generator $A \leftrightarrow B \leftrightarrow C$, we can trace the dependency between A and C even though they were not synchronized directly.

By periodically synchronizing the generators, each generator can then handle a portion of external traffic, thus the overall system can handle a larger amount of events to track at the cost of true time accuracy due to network latencies between the generators. A global order can still be achieved by picking some deterministic function to order any events that are within the synchronization window, such as by the value of the hash itself.

In Figure 5, the two generators insert each other's output state and record the operation. The color change indicates that data from the peer had modified the sequence. The generated hashes that are mixed into each stream are highlighted in bold.

The synchronization is transitive. $A \leftrightarrow B \leftrightarrow C$ There is a provable order of events between A and C through B.

Scaling in this way comes at the cost of availability. 10×1 gbps connections with availability of 0.999 would have $0.999^{10} = 0.99$ availability.

4.5 Consistency

Users are expected to be able to enforce consistency of the generated sequence and make it resistant to attacks by inserting the last observed output of the sequence they consider valid into their input.

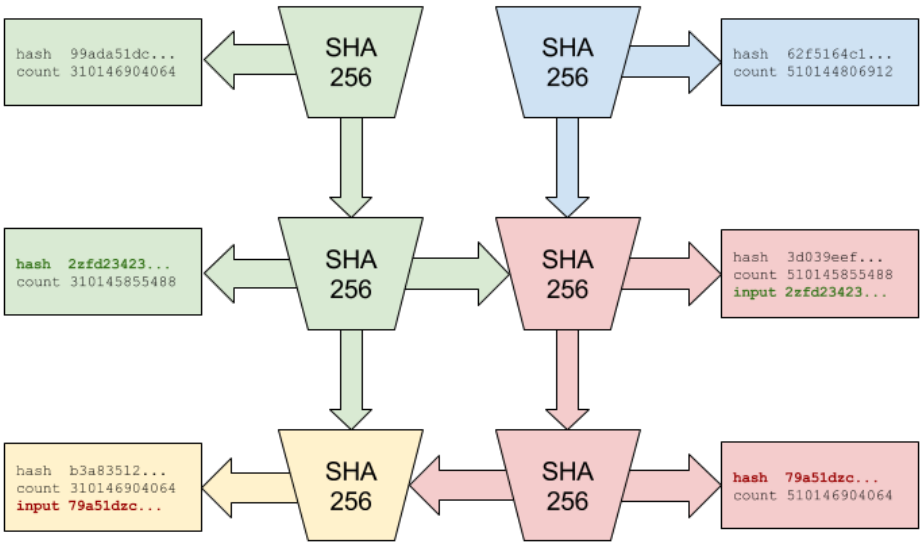


Figure 5: Two generators synchronizing

PoH Sequence A		
Index	Data	Output Hash
10		`hash10a`
20	Event1	`hash20a`
30	Event2	`hash30a`
40	Event3	`hash40a`

PoH Hidden Sequence B		
Index	Data	Output Hash
10		`hash10b`
20	Event3	`hash20b`
30	Event2	`hash30b`
40	Event1	`hash40b`

A malicious PoH generator could produce a second hidden sequence with the events in reverse order, if it has access to all the events at once, or is able to generate a faster hidden sequence.

To prevent this attack, each client-generated Event should contain within itself the latest hash that the client observed from what it considers to be a valid sequence. So when a client creates the ”Event1” data, they should append the last hash they have observed.

PoH Sequence A

Index	Data	Output Hash
10		`hash10a`
20	Event1 = append(event1 data, `hash10a`)	`hash20a`
30	Event2 = append(event2 data, `hash20a`)	`hash30a`
40	Event3 = append(event3 data, `hash30a`)	`hash40a`

When the sequence is published, Event3 would be referencing hash30a, and if it's not in the sequence prior to this Event, the consumers of the sequence know that it's an invalid sequence. The partial reordering attack would then be limited to the number of hashes produced while the client has observed an event and when the event was entered. Clients should then be able to write software that does not assume the order is correct for the short period of hashes between the last observed and inserted hash.

To prevent a malicious PoH generator from rewriting the client Event hashes, the clients can submit a signature of the event data and the last observed hash instead of just the data.

PoH Sequence A

Index	Data	Output Hash
10		hash10a
20	Event1 = sign(append(event1 data, hash10a), Client Private Key)	hash20a
30	Event2 = sign(append(event2 data, hash20a), Client Private Key)	hash30a
40	Event3 = sign(append(event3 data, hash30a), Client Private Key)	hash40a

Verification of this data requires a signature verification, and a lookup of the hash in the sequence of hashes prior to this one.
Verify:

```
(Signature, PublicKey, hash30a, event3 data) = Event3
Verify(Signature, PublicKey, Event3)
Lookup(hash30a, PoHSequence)
```

In Figure 6, the user-supplied input is dependent on hash `0xdeadbeef…` existing in the generated sequence sometime before it's inserted. The blue

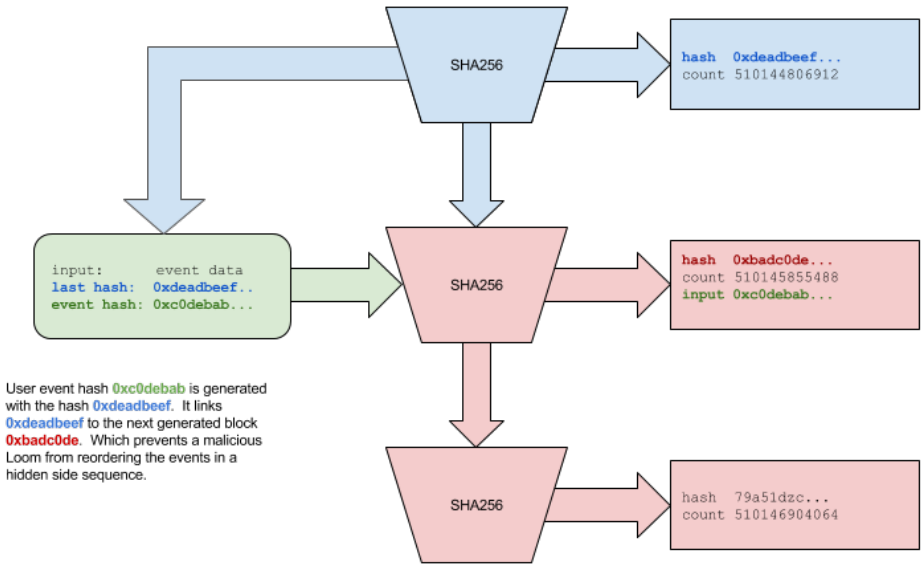


Figure 6: Input with a back reference.

top left arrow indicates that the client is referencing a previously produced hash. The client's message is only valid in a sequence that contains the hash `0xdeadbeef`…. The red color in the sequence indicates that the sequence has been modified by the clients data.

4.6 Overhead

4000 hashes per second would generate an additional 160 kilobytes of data, and would require access to a GPU with 4000 cores and roughly 0.25-0.75 milliseconds of time to verify.

4.7 Attacks

4.7.1 Reversal

Generating a reverse order would require an attacker to start the malicious sequence after the second event. This delay should allow any non malicious peer to peer nodes to communicate about the original order.

4.7.2 Speed

Having multiple generators may make deployment more resistant to attacks. One generator could be high bandwidth, and receive many events to mix into its sequence, another generator could be high speed low bandwidth that periodically mixes with the high bandwidth generator.

The high speed sequence would create a secondary sequence of data that an attacker would have to reverse.

4.7.3 Long Range Attacks

Long range attacks involve acquiring old discarded client Private Keys, and generating a falsified ledger [10]. Proof of History provides some protection against long range attacks. A malicious user that gains access to old private keys would have to recreate a historical record that takes as much time as the original one they are trying to forge. This would require access to a faster processor than the network is currently using, otherwise the attacker would never catch up in history length.

Additionally, a single source of time allows for construction of a simpler Proof of Replication (more on that in Section 6). Since the network is designed so that all participants in the network will rely on a single historical record of events.

PoRep and PoH together should provide a defense of both space and time against a forged ledger.

5 Proof of Stake Consensus

5.1 Description

This specific instance of Proof of Stake is designed for quick confirmation of the current sequence produced by the Proof of History generator, for voting and selecting the next Proof of History generator, and for punishing any misbehaving validators. This algorithm depends on messages eventually arriving to all participating nodes within a certain timeout.

5.2 Terminology

bonds Bonds are equivalent to a capital expense in Proof of Work. A miner buys hardware and electricity, and commits it to a single branch in a Proof of Work blockchain. A bond is coin that the validator commits as collateral while they are validating transactions.

slashing The proposed solution to the nothing at stake problem in Proof of Stake systems [7]. When a proof of voting for a different branch is published, that branch can destroy the validator's bond. This is an economic incentive designed to discourage validators from confirming multiple branches.

super majority A super majority is $\frac{2}{3}$rds of the validators weighted by their bonds. A super majority vote indicates that the network has reached consensus, and at least $\frac{1}{3}$rd of the network would have had to vote maliciously for this branch to be invalid. This would put the economic cost of an attack at $\frac{1}{3}$rd of the market cap of the coin.

5.3 Bonding

A bonding transaction takes a amount of coin and moves it to a bonding account under the user's identity. Coins in the bonding account cannot be spent and have to remain in the account until the user removes them. The user can only remove stale coins that have timed out. Bonds are valid after super majority of the current stakeholders have confirmed the sequence.

5.4 Voting

It is anticipated that the Proof of History generator will be able to publish a signature of the state at a predefined period. Each bonded identity must confirm that signature by publishing their own signed signature of the state. The vote is a simple yes vote, without a no. If super majority of the bonded identities have voted within a timeout, then this branch would be accepted as valid.

5.5 Unbonding

Missing N number of votes marks the coins as stale and no longer eligible for voting. The user can issue an unbonding transaction to remove them.

N is a dynamic value based on the ratio of stale to active votes. N increases as the number of stale votes increases. In an event of a large network partition, this allows the larger branch to recover faster than the smaller branch.

5.6 Elections

Election for a new PoH generator occur when the PoH generator failure is detected. The validator with the largest voting power, or highest public key address if there is a tie is picked as the new PoH generator.

A super majority of confirmations are required on the new sequence. If the new leader fails before a super majority confirmations are available, the next highest validator is selected, and a new set of confirmations is required.

To switch votes, a validator needs to vote at a higher PoH sequence counter, and the new vote needs to contain the votes it wants to switch. Otherwise the second vote will be slashable. Vote switching is expected to be designed so that it can only occur at a height that does not have a super majority.

Once a PoH generator is established, a Secondary can be elected to take over the transactional processing duties. If a Secondary exists, it will be considered as the next leader during a Primary failure.

The platform is designed so that the Secondary becomes Primary and lower rank generators are promoted if an exception is detected or on a predefined schedule.

5.7 Election Triggers

5.7.1 Forked Proof of History generator

PoH generators are designed with an identity that signs the generated sequence. A fork can only occur in case the PoH generator's identity has been compromised. A fork is detected because two different historical records have been published on the same PoH identity.

5.7.2 Runtime Exceptions

A hardware failure or a bug, or a intentional error in the PoH generator could cause it to generate an invalid state and publish a signature of the state that does not match the local validator's result. Validators will publish the correct signature via gossip and this event would trigger a new round of elections. Any validators who accept an invalid state will have their bonds slashed.

5.7.3 Network Timeouts

A network timeout would trigger an election.

5.8 Slashing

Slashing occurs when a validator votes two separate sequences. A proof of malicious vote will remove the bonded coins from circulation and add them to the mining pool.

A vote that includes a previous vote on a contending sequence is not eligible as proof of malicious voting. Instead of slashing the bonds, this vote removes the currently cast vote on the contending sequence.

Slashing also occurs if a vote is cast for an invalid hash generated by the PoH generator. The generator is expected to randomly generate an invalid state, which would trigger a fallback to Secondary.

5.9 Secondary Elections

Secondary and lower ranked Proof of History generators can be proposed and approved. A proposal is cast on the primary generator's sequence. The proposal contains a timeout, if the motion is approved by a super majority of the vote before the timeout, the Secondary is considered elected, and will take over duties as scheduled. Primary can do a soft handover to Secondary by inserting a message into the generated sequence indicating that a handover will occur, or inserting an invalid state and forcing the network to fallback to Secondary.

If a Secondary is elected, and the primary fails, the Secondary will be considered as the first fallback during an election.

5.10 Availability

CAP systems that deal with partitions have to pick Consistency or Availability. Our approach eventually picks Availability, but because we have an objective measure of time, Consistency is picked with reasonable human timeouts.

Proof of Stake verifiers lock up some amount of coin in a "stake", which allows them to vote for a particular set of transactions. Locking up coin is a transaction that is entered into a PoH stream, just like any other transaction. To vote, a PoS verifier has to sign the hash of the state, as it was computed after processing all the transactions to a specific position in the PoH ledger. This vote is also entered as a transaction into the PoH stream. Looking at the PoH ledger, we can then infer how much time passed between each vote, and if a partition occurs, for how long each verifier has been unavailable.

To deal with partitions with reasonable human timeframes, we propose a dynamic approach to "unstake" unavailable verifiers. When the number of verifiers is high and above $\frac{2}{3}$, the "unstaking" process can be fast. The number of hashes that must be generated into the ledger is low before the unavailable verifiers stake is fully unstaked and they are no longer counted for consensus. When the number of verifiers is below $\frac{2}{3}$rds but above $\frac{1}{2}$, the unstaking timer is slower, requiring a larger number of hashes to be generated before the missing verifiers are unstaked. In a large partition, like a partition that is missing $\frac{1}{2}$ or more of the verifiers, the unstaking process is very very slow. Transactions can still be entered into the stream, and verifiers can still vote, but full $\frac{2}{3}$rds consensus will not be achieved until a very large amount of hashes have been generated and the unavailable verifiers have been unstaked. The difference in time for a network to regain liveness allows us as customers of the network human timeframes to pick a partition that we want to continue using.

5.11 Recovery

In the system we propose, the ledger can be fully recovered from any failure. That means, anyone in the world can pick any random spot in the ledger and create a valid fork by appending newly generated hashes and transactions. If all the verifiers are missing from this fork, it would take a very very long time for any additional bonds to become valid and for this branch to achieve $\frac{2}{3}$rds super majority consensus. So full recovery with zero available validators

would require a very large amount of hashes to be appended to the ledger, and only after all the unavailable validators have been unstaked will any new bonds be able to validate the ledger.

5.12 Finality

PoH allows verifiers of the network to observe what happened in the past with some degree of certainty of the time of those events. As the PoH generator is producing a stream of messages, all the verifiers are required to submit their signatures of the state within 500ms. This number can be reduced further depending on network conditions. Since each verification is entered into the stream, everyone in the network can validate that every verifier submitted their votes within the required timeout without actually observing the voting directly.

5.13 Attacks

5.13.1 Tragedy of Commons

The PoS verifiers simply confirm the state hash generated by the PoH generator. There is an economic incentive for them to do no work and simply approve every generated state hash. To avoid this condition, the PoH generator should inject an invalid hash at a random interval. Any voters for this hash should be slashed. When the hash is generated, the network should immediately promote the Secondary elected PoH generator.

Each verifier is required to respond within a small timeout - 500ms for example. The timeout should be set low enough that a malicious verifier has a low probability of observing another verifiers vote and getting their votes into the stream fast enough.

5.13.2 Collusion with the PoH generator

A verifier that is colluding with the PoH generator would know in advance when the invalid hash is going to be produced and not vote for it. This scenario is really no different than the PoH identity having a larger verifier stake. The PoH generator still has to do all the work to produce the state hash.

5.13.3 Censorship

Censorship or denial of service could occur when a $\frac{1}{3}$rd of the bond holders refuse to validate any sequences with new bonds. The protocol can defend against this form of attack by dynamically adjusting how fast bonds become stale. In the event of a denial of service, the larger partition will be designed to fork and censor the Byzantine bond holders. The larger network will recover as the Byzantine bonds become stale with time. The smaller Byzantine partition would not be able to move forward for a longer period of time.

The algorithm would work as follows. A majority of the network would elect a new Leader. The Leader would then censor the Byzantine bond holders from participating. Proof of History generator would have to continue generating a sequence, to prove the passage of time, until enough Byzantine bonds have become stale so the bigger network has a super majority. The rate at which bonds become stale would be dynamically based on what percentage of bonds are active. So the Byzantine minority fork of the network would have to wait much longer than the majority fork to recover a super majority. Once a super majority has been established, slashing could be used to permanently punish the Byzantine bond holders.

5.13.4 Long Range Attacks

PoH provides a natural defense against long range attacks. Recovering the ledger from any point in the past would require the attacker to overtake the valid ledger in time by outpacing the speed of the PoH generator.

The consensus protocol provides a second layer of defense, as any attack would have to take longer than the time it takes to unstake all the valid validators. It also creates an availability "gap" in the history of the ledger. When comparing two ledgers of the same height, the one with the smallest maximum partition can be objectively considered as valid.

5.13.5 ASIC Attacks

Two opportunities for ASIC attacks exist in this protocol - during partition, and cheating timeouts in Finality.

For ASIC attacks during Partitions, the Rate at which bonds are unstaked is non-linear, and for networks with large partitions the rate is orders of magnitude slower than expected gains from an ASIC attack.

For ASIC attacks during Finality, the vulnerability allows for byzantine validators who have a bonded stake to wait for confirmations from other nodes and inject their votes with a collaborating PoH generator. The PoH generator can then use its faster ASIC to generate 500ms worth of hashes in less time, and allow for network communication between PoH generator and the collaborating nodes. But, if the PoH generator is also byzantine, there is no reason why the byzantine generator wouldn't have communicated the exact counter when they expect to insert the failure. This scenario is no different than a PoH generator and all the collaborators sharing the same identity vs. having a single combined stake and only using 1 set of hardware.

6 Streaming Proof of Replication

6.1 Description

Filecoin proposed a version of Proof of Replication [6]. The goal of this version is to have fast and streaming verifications of Proof of Replication, which are enabled by keeping track of time in a Proof of History generated sequence. Replication is not used as a consensus algorithm, but is a useful tool to account for the cost of storing the blockchain history or state at a high availability.

6.2 Algorithm

As shown in Figure 7 CBC encryption encrypts each block of data in sequence, using the previously encrypted block to XOR the input data.

Each replication identity generates a key by signing a hash that has been generated via a Proof of History sequence. This ties the key to a replicator's identity, and to a specific Proof of History sequence. Only specific hashes can be selected. (See Section 6.5 on Hash Selection)

The data set is fully encrypted block by block. Then to generate a proof, the key is used to seed a pseudorandom number generator that selects a random 32 bytes from each block.

A merkle hash is computed with the selected PoH hash prepended to the each slice.

The root is published, along with the key, and the selected hash that was generated. The replication node is required to publish another proof

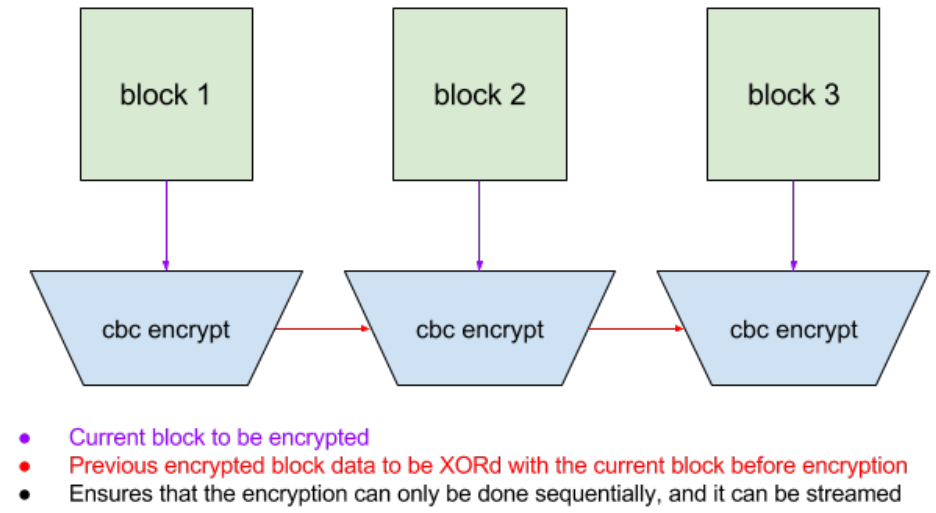


Figure 7: Sequential CBC encryption

in N hashes as they are generated by Proof of History generator, where N is approximately $\frac{1}{2}$ the time it takes to encrypt the data. The Proof of History generator will publish specific hashes for Proof of Replication at predefined periods. The replicator node must select the next published hash for generating the proof. Again, the hash is signed, and random slices are selected from the blocks to create the merkle root.

After a period of N proofs, the data is re-encrypted with a new CBC key.

6.3 Verification

With N cores, each core can stream encryption for each identity. Total space required is $2blocks * Ncores$, since the previous encrypted block is necessary to generate the next one. Each core can then be used to generate all the proofs that were derived from the current encrypted block.

Total time to verify proofs is expected to be equal to the time it takes to encrypt. The proofs themselves consume few random bytes from the block, so the amount of data to hash is significantly lower than the encrypted block size. The number of replication identities that can be verified at the same time is equal to the number of available cores. Modern GPUs have 3500+ cores available to them, albeit at $\frac{1}{2}$-$\frac{1}{3}$ the clock speed of a CPU.

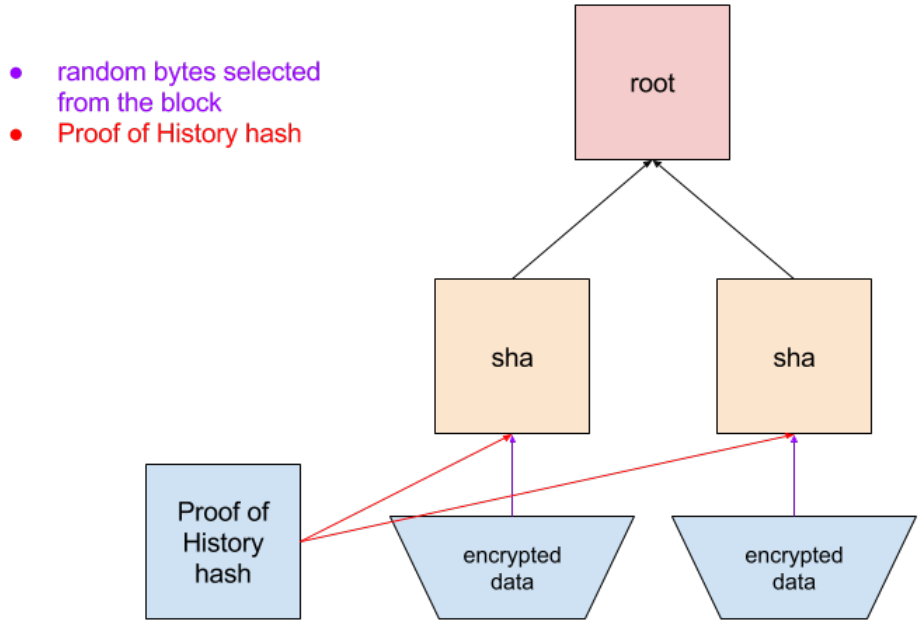


Figure 8: Fast Proof of Replication

6.4 Key Rotation

Without key rotation the same encrypted replication can generate cheap proofs for multiple Proof of History sequences. Keys are rotated periodically and each replication is re-encrypted with a new key that is tied to a unique Proof of History sequence.

Rotation needs to be slow enough that it's practical to verify replication proofs on GPU hardware, which is slower per core than CPUs.

6.5 Hash Selection

The Proof of History generator publishes a hash to be used by the entire network for encrypting Proofs of Replication, and for using as the pseudorandom number generator for byte selection in fast proofs.

The hash is published at a periodic counter that is roughly equal to $\frac{1}{2}$ the time it takes to encrypt the data set. Each replication identity must use the same hash, and use the signed result of the hash as the seed for byte

selection, or the encryption key.

The period that each replicator must provide a proof must be smaller than the encryption time. Otherwise the replicator can stream the encryption and delete it for each proof.

A malicious generator could inject data into the sequence prior to this hash to generate a specific hash. This attack is discussed more in 5.13.2.

6.6 Proof Validation

The Proof of History node is not expected to validate the submitted Proof of Replication proofs. It is expected to keep track of the number of pending and verified proofs submitted by the replicator's identity. A proof is expected to be verified when the replicator is able to sign the proof by a super majority of the validators in the network.

The verifications are collected by the replicator via p2p gossip network, and submitted as one packet that contains a super majority of the validators in the network. This packet verifies all the proofs prior to a specific hash generated by the Proof of History sequence, and can contain multiple replicator identities at once.

6.7 Attacks

6.7.1 Spam

A malicious user could create many replicator identities and spam the network with bad proofs. To facilitate faster verification, nodes are required to provide the encrypted data and the entire merkle tree to the rest of the network when they request verification.

The Proof of Replication that is designed in this paper allows for cheap verification of any additional proofs, as they take no additional space. But each identity would consume 1 core of encryption time. The replication target should be set to a maximum size of readily available cores. Modern GPUs ship with 3500+ cores.

6.7.2 Partial Erasure

A replicator node could attempt to partially erase some of the data to avoid storing the entire state. The number of proofs and the randomness of the seed should make this attack difficult.

For example, a user storing 1 terabyte of data erases a single byte from each 1 megabyte block. A single proof that samples 1 byte out of every megabyte would have a likelihood of collision with any erased byte $1 - (1 - 1/1,000,0000)^{1,000,000} = 0.63$. After 5 proofs the likelihood is 0.99.

6.7.3 Collusion with PoH generator

The signed hash is expected to be used to seed the sample. If a replicator could select a specific hash in advance then the replicator could erase all bytes that are not going to be sampled.

A replicator identity that is colluding with the Proof of History generator could inject a specific transaction at the end of the sequence before the predefined hash for random byte selection is generated. With enough cores, an attacker could generate a hash that is preferable to the replicator's identity.

This attack could only benefit a single replicator identity. Since all the identities have to use the same exact hash that is cryptographically signed with ECDSA (or equivalent), the resulting signature is unique for each replicator identity, and collision resistant. A single replicator identity would only have marginal gains.

6.7.4 Denial of Service

The cost of adding an additional replicator identity is expected to be equal to the cost of storage. The cost of adding extra computational capacity to verify all the replicator identities is expected to be equal to the cost of a CPU or GPU core per replication identity.

This creates an opportunity for a denial of service attack on the network by creating a large number of valid replicator identities.

To limit this attack, the consensus protocol chosen for the network can select a replication target, and award the replication proofs that meet the desired characteristics, like availability on the network, bandwidth, geolocation etc...

6.7.5 Tragedy of Commons

The PoS verifiers could simply confirm PoRep without doing any work. The economic incentives should be lined up with the PoS verifiers to do work, i.e. splitting the mining payout between the PoS verifiers and the PoRep replication nodes.

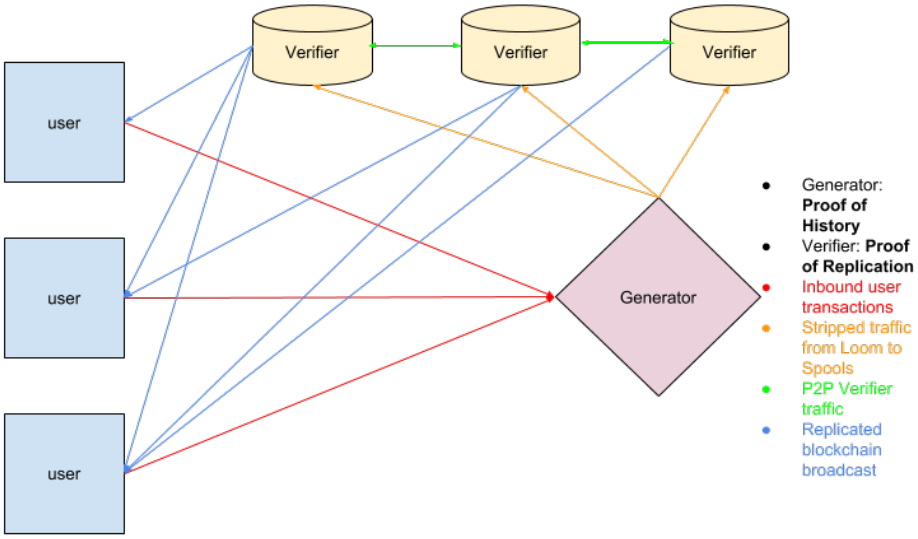


Figure 9: System Architecture

To further avoid this scenario, the PoRep verifiers can submit false proofs a small percentage of the time. They can prove the proof is false by providing the function that generated the false data. Any PoS verifier that confirmed a false proof would be slashed.

7 System Architecture

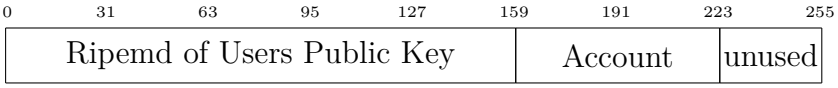
7.1 Components

7.1.1 Leader, Proof of History generator

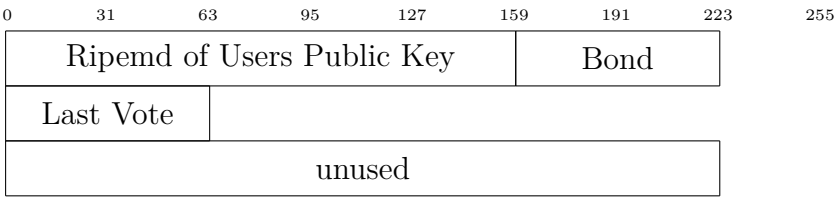
The Leader is an elected Proof of History generator. It consumes arbitrary user transactions and outputs a Proof of History sequence of all the transactions that guarantee a unique global order in the system. After each batch of transactions the Leader outputs a signature of the state that is the result of running the transactions in that order. This signature is signed with the identity of the Leader.

7.1.2 State

The state is a naive hash table indexed by the user's address. Each cell contains the full user's address and the memory required for this computation. For example,
the Transaction table contains:



for a total of 32 bytes.
The Proof of Stake bond's table contains:



for a total of 64 bytes.

7.1.3 Verifier, State Replication

The Verifier nodes replicate and provide high availability of the blockchain state. The replication target is selected by the consensus algorithm, and the validators in the consensus algorithm select and vote the Proof of Replication nodes they approve of based on off-chain defined criteria.

The network could be configured with a minimum Proof of Stake bond size, and a requirement for a single replicator identity per bond.

7.1.4 Validators

These nodes are consuming bandwidth from Verifiers. They are virtual nodes, and can run on the same machines as the Verifiers or the Leader, or on separate machines that are specific to the consensus algorithm configured for this network.

7.2 Network Limits

The leader is expected to be able to take incoming user packets, order them in the most efficient way possible, and sequence them into a Proof of History

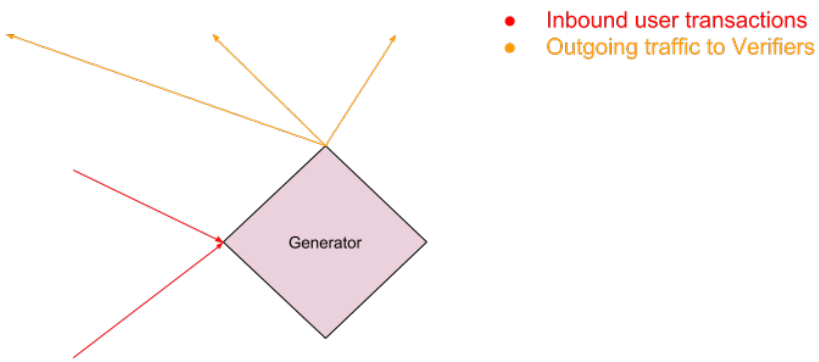
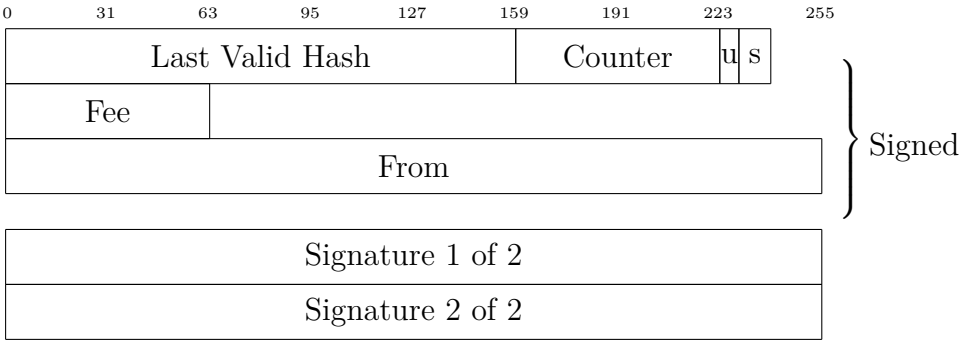


Figure 10: Generator network limits

sequence that is published to downstream Verifiers. Efficiency is based on memory access patterns of the transactions, so the transactions are ordered to minimize faults and to maximize prefetching.

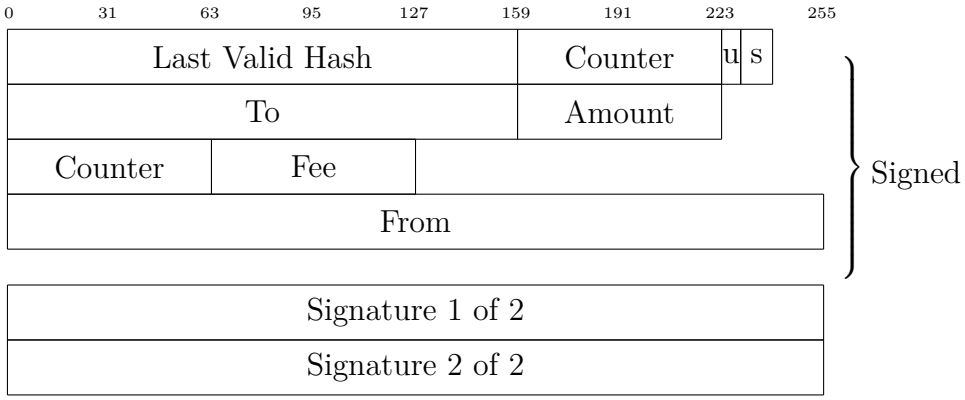
Incoming packet format:



Size $20 + 8 + 16 + 8 + 32 + 32 + 32 = 148$ bytes.

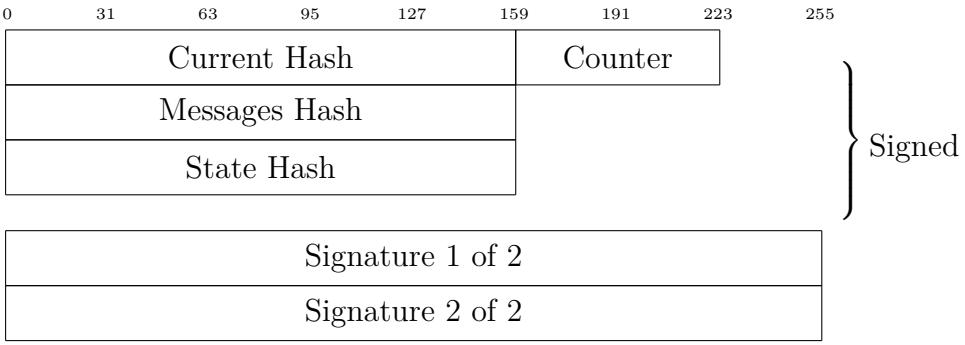
The minimal payload that can be supported would be 1 destination account with a minimum size of 176 bytes.

With payload:



With payload the minimum size: 176 bytes

The Proof of History sequence packet contains the current hash, counter, and the hash of all the new messages added to the PoH sequence and the state signature after processing all the messages. This packet is sent once every N messages are broadcast.
Proof of History packet:



Minimum size of the output packet is: 132 bytes

On a 1gbps network connection the maximum number of transactions possible is 1 gigabit per second / 176 bytes = 710k tps max. Some loss $(1 - 4\%)$ is expected due to Ethernet framing. The spare capacity over

the target amount for the network can be used to increase availability by coding the output with Reed-Solomon codes, and striping it to the available downstream Verifiers.

7.3 Computational Limits

Each transaction requires a digest verification. This operation does not use any memory outside of the transaction message itself and can be parallelized independently. Thus throughput is expected to be limited by the number of cores available on the system.

GPU based ECDSA verification servers have had experimental results of 900k operations per second [9].

7.4 Memory Limits

A naive implementation of the state as a 50% full hashtable with 32 byte entries for each account, would theoretically fit 10 billion accounts into 640GB. Steady state random access to this table is measured at $1.1 * 10^7$ writes or reads per second. Based on 2 reads and two writes per transaction, memory throughput can handle 2.75m transactions per second. This was measured on an Amazon Web Services 1TB x1.16xlarge instance.

7.5 High Performance Smart Contracts

Smart contracts are a generalized form of transactions. These are programs that run on each node and modify the state. This design leverages extended Berkeley Packet Filter bytecode, which is fast and easy to analyze, and JIT bytecode as the smart contracts language.

One of its main advantages is a zero cost Foreign Function Interface. Intrinsics, or functions that are implemented on the platform directly, are callable by programs. Calling the intrinsics suspends that program and schedules the intrinsic on a high performance server. Intrinsics are batched together to execute in parallel on the GPU.

In the above example, two different user programs call the same intrinsic. Each program is suspended until the batch execution of the intrinsics is complete. An example intrinsic is ECDSA verification. Batching these calls to execute on the GPU can increase throughput by thousands of times.

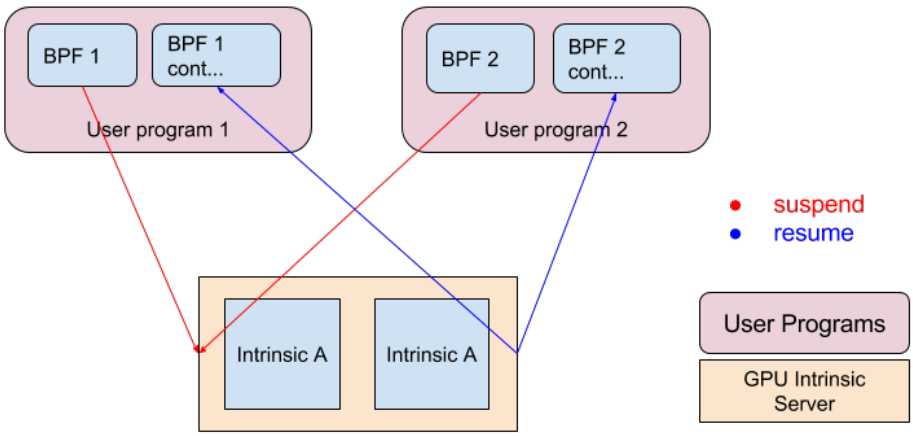


Figure 11: Executing BPF programs.

This trampoline requires no native operating system thread context switches, since the BPF bytecode has a well defined context for all the memory that it is using.

eBPF backend has been included in LLVM since 2015, so any LLVM frontend language can be used to write smart contracts. It's been in the Linux kernel since 2015, and the first iterations of the bytecode have been around since 1992. A single pass can check eBPF for correctness, ascertain its runtime and memory requirements and convert it to x86 instructions.

References

[1] Liskov, Practical use of Clocks
`http://www.dainf.cefetpr.br/ tacla/SDII/PracticalUseOfClocks.pdf`

[2] Google Spanner TrueTime consistency
`https://cloud.google.com/spanner/docs/true-time-external-consistency`

[3] Solving Agreement with Ordering Oracles
`http://www.inf.usi.ch/faculty/pedone/Paper/2002/2002EDCCb.pdf`

[4] Tendermint: Consensus without Mining
`https://tendermint.com/static/docs/tendermint.pdf`

[5] Hedera: A Governing Council & Public Hashgraph Network
https://s3.amazonaws.com/hedera-hashgraph/hh-whitepaper-v1.0-180313.pdf

[6] Filecoin, proof of replication,
https://filecoin.io/proof-of-replication.pdf

[7] Slasher, A punative Proof of Stake algorithm
https://blog.ethereum.org/2014/01/15/slasher-a-punitive-proof-of-stake-algorithm/

[8] BitShares Delegated Proof of Stake
https://github.com/BitShares/bitshares/wiki/Delegated-Proof-of-Stake

[9] An Efficient Elliptic Curve Cryptography Signature Server With GPU Acceleration
http://ieeexplore.ieee.org/document/7555336/

[10] Casper the Friendly Finality Gadget
https://arxiv.org/pdf/1710.09437.pdf