SAPIENZA
UNIVERSITÀ DI ROMA

SCIENCE AND TECHNOLOGY

Solana Priority Fees Crisis

SAPIENZA SCHOOL OF ADVANCED STUDIES

Student:
Valerio Massimo Camaiani
1935012

Advisor:
Giovanni Farina
Tutor:
Emanuele Rodolà

Academic Year 2023/2024

Contents

1 Introduction **2**

2 Crisis Overview **3**

2.1 Non-technical introduction . . . 3

2.2 Aftermath . . . 4

2.3 Technical Cause and Solution . . . 5

3 The Solana Blockchain **8**

3.1 Introduction . . . 8

3.2 History Of Solana . . . 8

3.2.1 Development Timeline: Tracing Key Milestones and Updates . . 8

3.2.2 Growth and Adoption: Charting Solana's Rise in Popularity . . 9

3.3 Purpose and Intents . . . 10

3.4 Technical Overview . . . 12

3.4.1 Consensus Mechanism: Proof of History (PoH) . . . 12

3.4.2 High Throughput and Low Latency Focus . . . 12

3.4.3 Additional features . . . 13

3.5 Challenges and Issues . . . 13

4 Priority Fees **16**

4.1 Overview . . . 16

4.2 The failure . . . 17

4.3 The fix . . . 18

5 Scheduling **19**

5.1 Legacy scheduler . . . 19

5.1.1 Local queue . . . 21

5.1.2 Replicating the prioritization jitter . . . 21

5.2 Central scheduler . . . 25

5.2.1 Proving the prioritization consistency . . . 27

6 Conclusions **29**

7 Appendix **30**

References **32**

1 Introduction

Solana is an extremely promising blockchain that has seen incredible growth thanks to its high throughput, low transaction costs, and innovative consensus mechanisms. Founded in 2018, Solana quickly gained traction in the blockchain space, attracting a wide range of decentralized applications (DApps), decentralized finance (DeFi) projects, and non-fungible token (NFT) platforms. Its unique Proof of History (PoH) consensus algorithm and advanced architectural features enable Solana to process thousands of transactions per second while maintaining a high level of security and decentralization.

However, the network has often been subject to reliability issues. Most recently, in April 2024, Solana faced a severe congestion crisis triggered by an unprecedented surge in low value coin trading and bot-generated spam transactions. This event highlighted significant vulnerabilities in Solana's infrastructure, particularly its ability to handle sudden spikes in network activity.

The April 2024 crisis affected the trust of stakeholders in Solana and caused a sharp decline in the price of SOL, Solana's native token. Transaction failures, network slowdowns, and a high volume of spam transactions led to widespread dissatisfaction among users and investors. One of the major shortcomings revealed during the crisis was the unreliable implementation of priority fees. Despite the system's intent to allow users to expedite transactions by paying higher fees in case of high congestion, it failed in the very condition it was designed to mitigate.

In this thesis, we will analyze all the above-mentioned themes, providing a comprehensive overview of Solana's development, the April 2024 crisis, and its impact on stakeholder trust and network performance. The core of this work involves a technical deep dive into Solana's different schedulers, explaining their inner workings to replicate the problem and then analyzing the solutions implemented to address these critical issues. By exploring these themes in depth, this thesis aims to contribute valuable insights into the management and scalability of blockchain networks, ensuring their resilience and efficiency in the face of future challenges.

Through a detailed and cohesive analysis, this thesis seeks to provide a deep understanding of Solana's response to the April 2024 crisis, the technical improvements made, and the strategic initiatives undertaken to ensure the network's resilience and efficiency in the face of future challenges.

2 Crisis Overview

2.1 Non-technical introduction

In April 2024, the Solana network faced severe congestion largely due to the unprecedented surge in meme coin trading and bot-generated spam transactions. This period highlighted significant vulnerabilities in Solana's infrastructure, particularly its ability to manage sudden spikes in network activity.

The congestion started the first days of April and was primarily driven by the popularity of meme coins such as Dogwifhat (WIF), Bonk (BONK), and Book of Meme (BOME). These tokens, often created around childish and humorous concepts, drew large numbers of traders. These traders were attracted by Solana's high throughput and low transaction fees, which made it an ideal platform for quick speculative trading. However, this also made the network a target for bots programmed to spam the network with a high volume of low-cost transactions, hoping to capitalize on the congestion by sneaking in transactions amidst the chaos. This flood of activity created bottlenecks, significantly hampering the network's performance and leading to a failure rate as high as 75% during peak times.

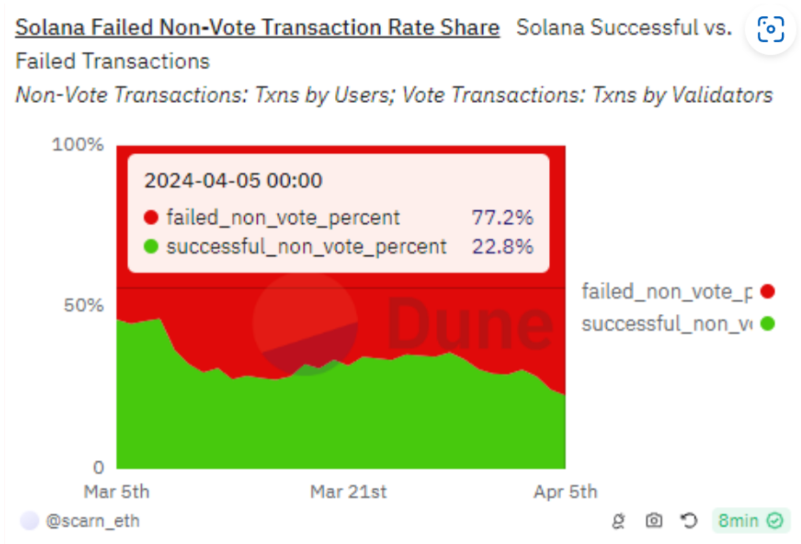


Figure 1: 75% failed transactions

Moreover, the congestion was compounded by speculative activities such as token presales, where developers raised funds for projects that often had not yet materialized. This speculative bubble raised concerns about potential "rug pulls", where developers could abruptly withdraw from projects and abscond with the funds, leaving investors with worthless tokens. The spike in network activity from these presales and speculative trades exacerbated the already strained network, leading to widespread transaction delays and failures. [1]

Anatoly Yakovenko, Solana's co-founder, recognized that while the meme coin craze brought increased attention and liquidity to Solana, it also exposed the network to new types of vulnerabilities. Many of the transaction failures were attributed to the overwhelming spam from bots exploiting the low fee structure to flood the network. Yakovenko noted that this ordeal served as a significant stress test, providing crucial insights into the network's scalability challenges and areas needing improvement. [2]

To address the immediate congestion issues, Solana had implemented a system of priority fees, where users could pay extra to expedite their transactions. However, during the crisis, this system did not perform as expected. Despite users paying higher fees, the sheer volume of transactions continued to saturate the network, rendering the priority fee system ineffective. This situation highlighted the need for more robust mechanisms to manage network load and prioritize transactions during periods of peak demand effectively.

The April 2024 crisis not only tested Solana's capacity to handle high transaction volumes but also sparked a broader discussion within the community about the network's resilience and the critical need for implementing effective measures to manage congestion. [3] [4]

2.2 Aftermath

The crisis had a devastating effect on the Solana ecosystem failing stakeholders trust. The sharp 56% drop in the price of SOL from $193 to $123 starkly illustrated the market's reaction to unresolved network issues and declining on-chain activity.These troubles were further highlighted by a rollercoaster price recovery and subsequent fallback, underscoring the network's volatility during this period.

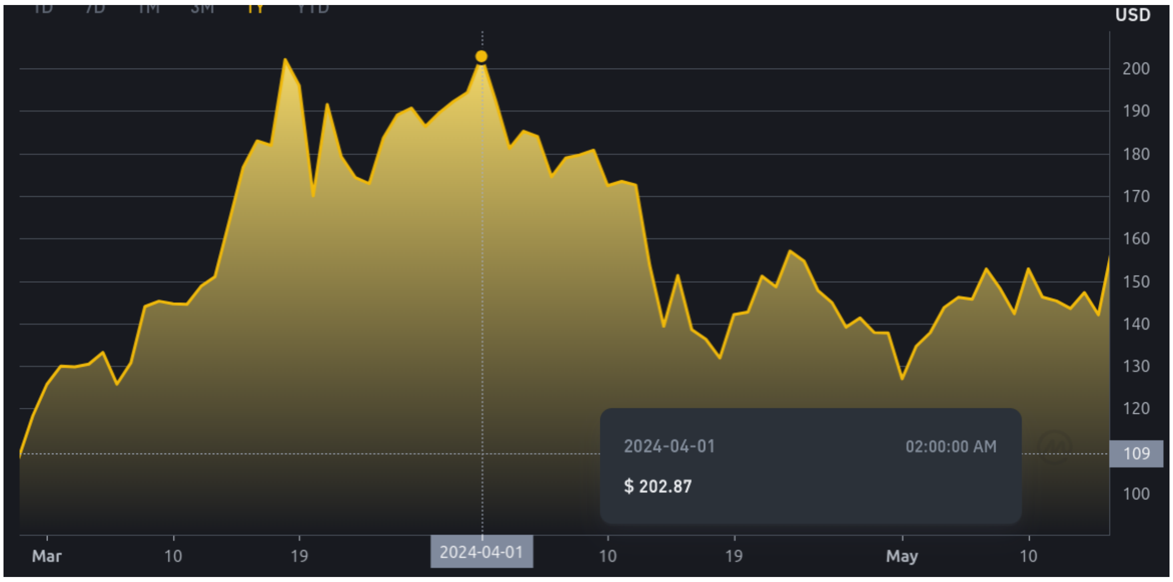


Figure 2: SOL drop after 4th April

The NFT sector on Solana mirrored this volatility, with a complete wipeout of sales volume growth compared to the previous month, plummeting to \$153.4 million. This drastic fall was a direct of the slowdown in network transactions due to the ongoing congestion issues but also reflection of diminished user confidence.

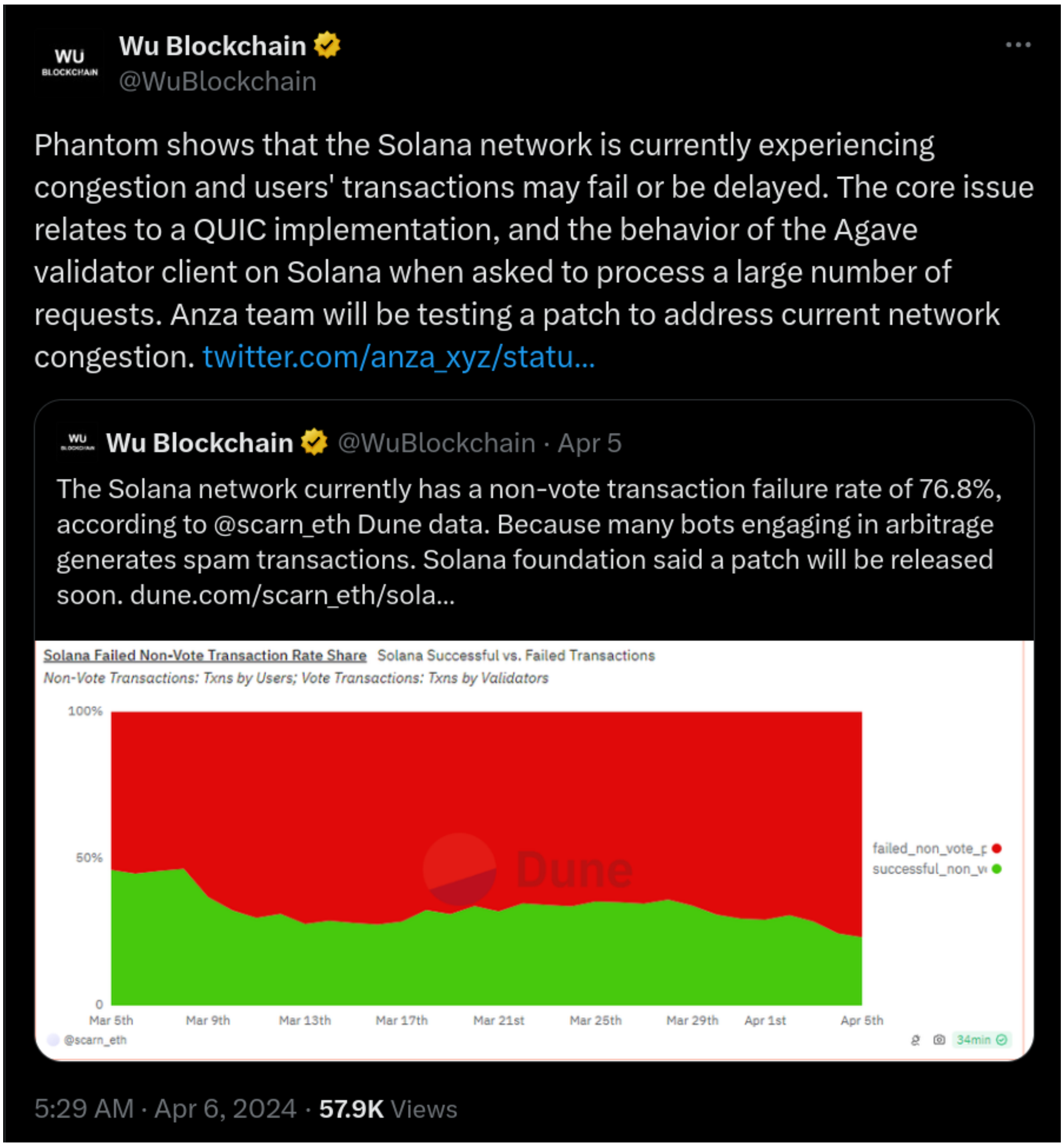
Interestingly, despite these challenges, there was a notable surge in institutional investment in SOL. CoinShares reported that nearly 15% of surveyed investors now held SOL, indicating a robust institutional faith in the potential resilience and future growth of the Solana network. This period also saw FTX's bankruptcy estate liquidate 30 million SOL tokens, generating about \$1.9 billion, intended to compensate defrauded customers—a move that significantly impacted SOL's market dynamics. [5]

2.3 Technical Cause and Solution

The root of the congestion can be traced to the implementation of the QUIC protocol within Solana's architecture. QUIC, initially developed by Google, is designed to make internet communications faster and more reliable was not fully optimized for Solana's needs [6]. Austin Federa, the strategic director at the Solana Foundation, acknowledged that the rapid increase in demand for Solana's services transformed what was once a manageable level of technical debt into an insurmountable issue almost overnight. He indicated that the network's stack was scheduled for a comprehensive rewrite later in the year to address these inefficiencies comprehensively. The Agave client's implementation compounded these problems. [7]

Matt Sorg, Tech and Product Lead at Solana, claimed that many transaction failures were not indicative of fundamental flaws within the network but were instead designed as protections against executing transactions under problematic conditions.

[8][9]



Wu Blockchain @WuBlockchain

Phantom shows that the Solana network is currently experiencing congestion and users' transactions may fail or be delayed. The core issue relates to a QUIC implementation, and the behavior of the Agave validator client on Solana when asked to process a large number of requests. Anza team will be testing a patch to address current network congestion. twitter.com/anza_xyz/statu...

Wu Blockchain @WuBlockchain · Apr 5

The Solana network currently has a non-vote transaction failure rate of 76.8%, according to @scarn_eth Dune data. Because many bots engaging in arbitrage generates spam transactions. Solana foundation said a patch will be released soon. dune.com/scarn_eth/sola...

5:29 AM · Apr 6, 2024 · 57.9K Views

Solana Failed Non-Vote Transaction Rate Share
Solana Successful vs. Failed Transactions
Non-Vote Transactions: Txns by Users; Vote Transactions: Txns by Validators
100%
50%
0
Mar 5th
Mar 9th
Mar 13th
Mar 17th
Mar 21st
Mar 25th
Mar 29th
Apr 1st
Apr 5th
Dune
failed_non_vote_p
successful_non_v
@scarn_eth
34min

Figure 3: Post from WuBlockchain [10]

Solana's response to these challenges included several strategic updates aimed at addressing the congestion and improving the overall robustness of the network. On April 12, developers released update v1.17.31, introducing enhancements meant to alleviate congestion and optimize the handling of high-demand periods spurred by meme coin activities. Additionally, the network's validators approved a new system intended to expedite transaction processing, reflecting a proactive community striving for improvement and efficiency. These targeted actions were crucial in mitigating the immediate effects of the crisis and laying the groundwork for a more resilient network.

The real solution however was expected to be released with the launch of v1.18 [11] [12]

3 The Solana Blockchain

3.1 Introduction

Solana is a high-performance blockchain platform designed to deliver scalability, low latency, and low costs. Founded in 2018, Solana introduces a novel Proof of History (PoH) consensus algorithm to address the challenges of blockchain scalability and speed. Since its inception, Solana has rapidly grown to become one of the leading platforms in the blockchain space, attracting a wide range of decentralized applications (DApps), DeFi projects, and NFT platforms.

Solana's architecture is engineered to handle thousands of transactions per second while maintaining a high level of security and decentralization. Key innovations such as the Turbine block propagation protocol, Gulf Stream transaction forwarding, and Sealevel parallel smart contract runtime contribute to its high throughput and efficiency. Additionally, Solana's commitment to low transaction fees and environmental sustainability makes it a compelling choice for developers and users alike.

In this chapter, we will explore the history of Solana, its core objectives, technical architecture, and the challenges it faces. We will also examine the significant milestones in Solana's development, its growth in popularity, and the impact of broader market conditions on its evolution. This comprehensive overview will provide a deep understanding of what sets Solana apart in the rapidly evolving blockchain landscape.

3.2 History Of Solana

3.2.1 Development Timeline: Tracing Key Milestones and Updates

Solana's development began in 2017 when Anatoly Yakovenko, a former Qualcomm engineer, proposed Proof of History (PoH), a novel consensus algorithm aimed at solving blockchain scalability issues. In 2018, Yakovenko, along with Greg Fitzgerald, Stephen Akridge, and Raj Gokal, founded Solana Labs. By June 2018, they had developed a testnet capable of handling up to 250,000 transactions per second (TPS).

Throughout 2019, Solana made significant strides, with key releases like v0.11.0 and v0.14.0 improving network infrastructure and performance. The incentivized testnet event, "Tour de SOL", further engaged the validator community to enhance network robustness.

In March 2020, Solana launched its Mainnet Beta, demonstrating the ability to process up to 50,000 TPS, a major milestone distinguishing it from other blockchains. This period saw rapid ecosystem expansion, attracting numerous decentralized applications (DApps), DeFi projects, and NFT platforms due to Solana's high performance and low fees.

By 2022, Solana introduced Solana Pay and collaborated with Google Cloud, solidifying its presence in the blockchain space.

As of 2023, Solana's evolution includes initiatives like the launch of the web3-focused Android smartphone, Saga, and ongoing efforts to improve network performance and developer support.

3.2.2 Growth and Adoption: Charting Solana's Rise in Popularity

Since its inception, Solana has experienced remarkable growth and adoption, solidifying its position as a leading blockchain platform. Solana's journey to widespread adoption began with the launch of its Mainnet Beta in March 2020, which demonstrated its capability to handle high transaction volumes efficiently. The initial price of SOL during this launch was $0.9511, and it soared to an all-time high of $260.06 by November 2021 during the peak of the crypto bubble.

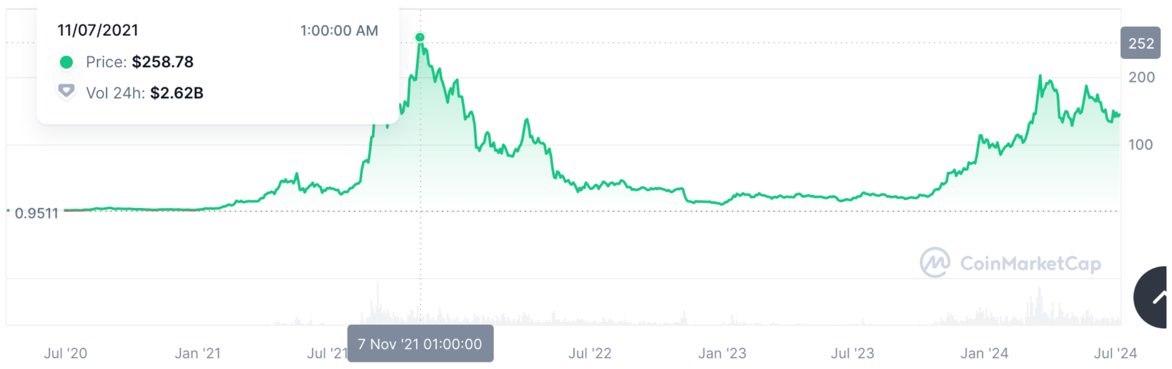


Figure 4: SOL price history

Several high-profile partnerships and integrations have significantly boosted Solana's visibility and credibility within the blockchain industry.

- Serum: A decentralized exchange (DEX) that leverages Solana's high-speed capabilities.
- Google Cloud: Integration with Solana to enable high-performance use cases and efficient data processing.
- Visa: Expansion of its USDC stablecoin pilot program to the Solana network, enhancing payment capabilities.
- Mastercard: Collaboration to secure transactions between web2 and web3 institutions.

Solana's developer ecosystem has flourished, with numerous hackathons and educational programs encouraging active participation. The platform has seen thousands of developers contributing to its growth through various initiatives, such as Grizzlython and Hyperdrive hackathons, which have witnessed record numbers of submissions. This vibrant developer community has been instrumental in expanding Solana's ecosystem.

- DeFi Projects: Solana's high throughput and low fees have attracted numerous DeFi projects, positioning it as a significant player in the space.
- NFT Marketplaces: Platforms like Solanart and projects like Degenerate Ape Academy have thrived on Solana, contributing to its growing popularity in the NFT ecosystem.

During the crypto market downturn in 2022, Solana's value fell substantially. This period saw SOL's price drop from its high of over \$200 in November 2021 to as low as \$33.31 by mid-2022. But, contrary to many other entities in the crypto space, Solana recovered and as of June 2024, Solana's market capitalization stands at approximately \$68.41 billion, with a circulating supply of 462.35 million SOL. The current price of SOL is around \$146.81, reflecting an 800.46% increase over the past year. Additionally, Solana processes about 25 million non-voting transactions daily, showcasing its capability to handle significant transaction volumes. The number of active wallets has surged, with a 58% increase in active wallets since the start of the year. [13] [14]

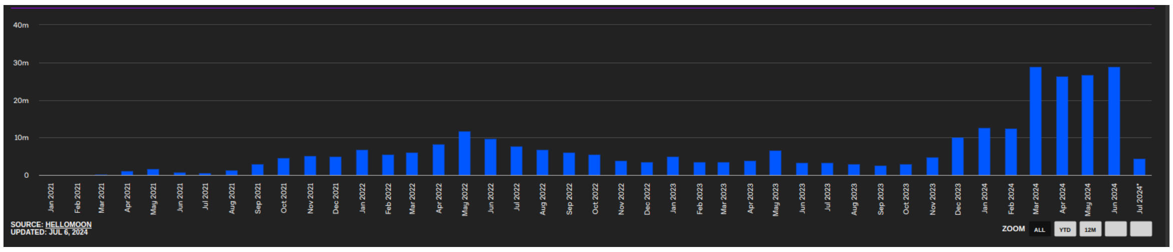


Figure 5: Number of new Wallets (monthly)

3.3 Purpose and Intents

Solana, was developed with several core objectives aimed at enhancing scalability, usability, and the overall blockchain experience for developers and users alike [15] [16].

High Throughput and Scalability One of Solana's primary goals is to achieve high throughput, allowing the network to process thousands of transactions per second. This is made possible through its unique Proof of History (PoH) mechanism, which creates a historical record that proves that an event has occurred at a specific moment in time. By integrating PoH with Proof of Stake (PoS), Solana can efficiently order and validate transactions, significantly increasing its scalability without compromising decentralization or security.

Low Latency and Fast Settlement

Solana aims to minimize transaction latency, achieving sub-second settlement times. Each transaction is confirmed in approximately 400 milliseconds, making it one of the fastest blockchains available. This rapid confirmation time enhances user experience by providing near-instantaneous transaction feedback, which is crucial for applications requiring real-time interaction.

Low Transaction Costs Another critical objective of Solana is to maintain low transaction fees. With a median fee of just \$0.00064 per transaction, Solana ensures that the cost of using the blockchain remains accessible to a wide range of users, promoting mass adoption. This low-cost environment is particularly beneficial for developers and businesses that need to handle a large volume of transactions without incurring prohibitive costs allowing for new usecases to be brought onchain.

Gas-Free Environment for Developers Solana's architecture is designed to provide a gas-free environment, reducing the financial burden on developers when creating and deploying decentralized applications (dApps). This allows for more innovation and experimentation within the ecosystem, as developers are not constrained by high operational costs.

Focus on DApp Scalability and Faster Iteration Solana prioritizes the scalability of decentralized applications, enabling them to support millions of users without performance bottlenecks. This scalability is achieved through its high throughput and low latency features. Solana's infrastructure is designed to accommodate the growth of dApps, also allowing developers to iterate and improve their applications more rapidly

Promoting Decentralization Solana is committed to maintaining a decentralized network by increasing the number of validator nodes and enhancing its censorship resistance. This is achieved by ensuring that no single entity can control or disrupt the network, providing a secure and reliable platform for all participants. More validators are being developed to allow nodes to chose which code to run, for example Firedancer has been recently released by JumpCrypto providing a faster alternative to the default SolanaLabs validator [17] [18].

Environmental Efficiency Solana also focuses on being environmentally friendly. By using a PoS mechanism alongside PoH, it significantly reduces its carbon footprint compared to traditional Proof of Work (PoW) blockchains. Each transaction on Solana consumes minimal energy, comparable to a few Google searches, making it a more sustainable option for blockchain technology.

3.4 Technical Overview

Solana's architecture is centered around optimizing transaction speeds while maintaining a high level of security and decentralization. The core components of Solana include its Proof of History (PoH) consensus mechanism, a high-speed synchronization engine, and an efficient transaction processing protocol.

3.4.1 Consensus Mechanism: Proof of History (PoH)

Proof of History (PoH) is a cryptographic clock designed to provide a way for nodes in a network to agree on the time and sequence of events with minimal interaction. Unlike traditional consensus methods, which require significant communication between nodes, PoH reduces this overhead. In PoH, transactions are hashed sequentially using the SHA-256 algorithm, creating a cryptographically secure and verifiable record of each event's timing and order. This is essential for maintaining the integrity and chronological accuracy of transactions on the blockchain.

Comparison: Traditional blockchains like Bitcoin use Proof of Work (PoW), requiring miners to solve complex computations, leading to slower transaction times and higher energy consumption. Ethereum transitioned in 2022 to Proof of Stake (PoS), which reduces energy usage but still involves considerable communication between validators. PoH streamlines this process by providing a pre-agreed timeline for transactions.

3.4.2 High Throughput and Low Latency Focus

Solana achieves high throughput and low latency through a combination of technologies designed to optimize speed and efficiency:

- **Turbine:** A block propagation protocol that breaks data into smaller packets, enabling faster transmission and reduced bandwidth requirements. Turbine ensures quick information dissemination across the network, akin to a peer-to-peer network optimization technique.

- **Gulf Stream:** A protocol for transaction forwarding that allows validators to execute transactions ahead of time, significantly reducing confirmation times. By pre-emptively forwarding transactions to the next set of validators, Gulf Stream helps maintain a steady flow of transaction processing. This is in contrast to Bitcoin's block confirmation time, which averages around 10 minutes.

- **Sealevel:** Solana's parallel smart contract runtime allows multiple contracts to run concurrently using a single state machine. Unlike Ethereum, which processes transactions sequentially due to its current infrastructure, Sealevel enables parallel processing, enhancing scalability and performance significantly.

3.4.3 Additional features

Optimized Data Storage: Archivers

Solana employs a decentralized data storage solution known as Archivers. These are lightweight nodes responsible for storing data from the blockchain network without participating in consensus. Archivers maintain the integrity and availability of historical data, ensuring the blockchain remains lightweight and efficient without sacrificing data accessibility or security.

Comparison: In contrast, traditional blockchain networks like Ethereum and Bitcoin require all nodes to store the entire blockchain history, leading to significant storage overhead. Solana's Archiver nodes offload this requirement, making the network more scalable.

Pipeline: Optimized Transaction Processing Unit

The Pipeline is Solana's transaction processing unit that optimizes the validation process by breaking it down into stages handled by different hardware components in parallel. This pipelined approach ensures transactions are processed quickly and efficiently, contributing to Solana's high throughput and low latency.

Comparison: Traditional blockchains often process transactions sequentially, leading to bottlenecks. Solana's Pipeline, similar to modern CPU design, allows for simultaneous processing of multiple transactions, enhancing overall performance.

Cloudbreak: Horizontal Scaling of Accounts

Cloudbreak is a data structure designed for horizontal scaling of accounts, allowing the system to read and write data concurrently across multiple SSDs. This ensures that the system can handle a vast number of accounts and transactions efficiently.

Comparison: Many blockchain systems struggle with horizontal scalability, often facing issues with data bottlenecks. Solana's Cloudbreak design mitigates these issues by enabling concurrent data access, thus enhancing the system's capacity and speed.

3.5 Challenges and Issues

Despite its impressive growth and technological advancements, Solana has encountered several challenges and issues that have impacted its reliability and user experience. These challenges can be broadly categorized into network stability, security concerns, and the effects of broader crypto market downturns.

Network Stability

One of the most significant issues Solana has faced is network stability. The platform has experienced multiple outages and network slowdowns since its launch, which have raised concerns about its reliability. Maintaining consistent uptime is critical for user trust and adoption, and Solana has struggled in this area.

Notable Outages:

- September 2021 Outage: Solana experienced a 17-hour outage caused by an overwhelming transaction load during an Initial DEX Offering (IDO) on the Raydium platform. The surge, peaking at 400,000 transactions per second, led to a denial of service and network halt. Validators could not reach consensus, necessitating a coordinated restart by the validator community.
- January 2022 Outage: Between January 6-12, 2022, Solana faced multiple partial outages and degraded performance due to a significant increase in high compute transactions. This overwhelmed the network's capacity, resulting in several instances of reduced performance and intermittent downtimes.
- February 2022 Outage: Solana faced another major outage in February 2022, lasting approximately 48 hours. A large-scale bot attack targeting the network's NFT minting activities caused an overwhelming influx of transactions, leading to network congestion and a halt.
- October 2022 Outage: On October 1, 2022, Solana suffered a six and a half-hour outage due to a bug in the network's code triggered by a misconfigured validator. This bug led to a fork in the network, causing validators to fall out of consensus and halting block production until the issue was resolved.
- February 2024 Outage: Solana experienced its 11th outage, halting block production for over 25 minutes. This incident once again raised questions about the network's stability and reliability [19] [20].

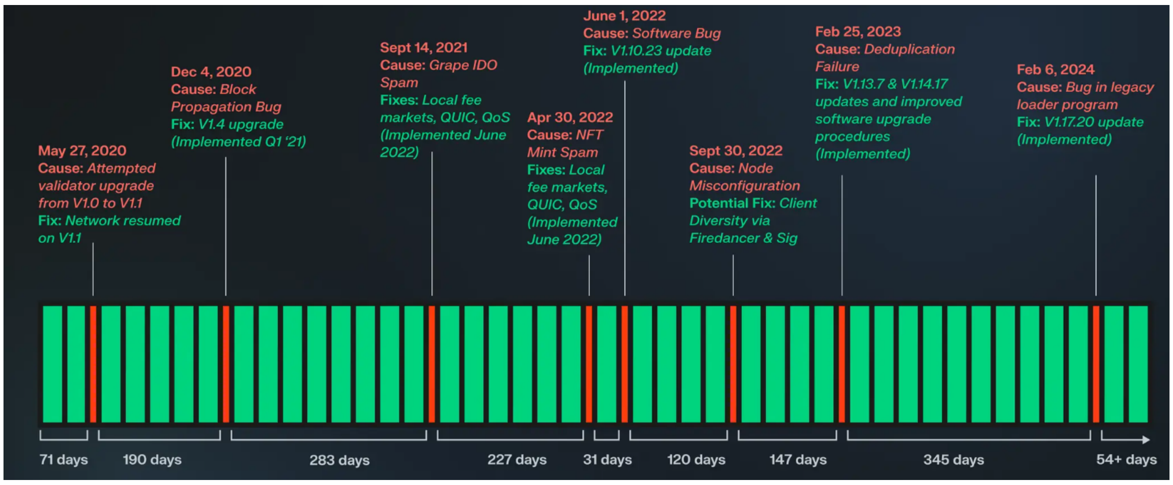


Figure 6: History of Solana outages [21]

Security Concerns

Solana has grappled with security issues, including vulnerabilities and attacks. Notable incidents like the Wormhole and Raydium protocol attacks have exposed weaknesses in its security infrastructure. These incidents underscore the need for continuous improvement in security measures to protect user assets and maintain confidence in the network.

Network Congestion

Given the very low fees network congestion is another persistent issue for Solana. High demand and traffic spikes, often caused by spam, have led to slower transaction processing times and increased failure rates. For instance, the April 2024 crisis we analyzed is one such example.

Impact of Crypto Market Downturns

The broader cryptocurrency market is known for its volatility, and significant downturns have affected nearly every asset in the space, including Solana. During major market corrections, the value of SOL, has often mirrored the broader market's declines. In early 2022, a significant drop in the overall cryptocurrency market led to a sharp decrease in SOL's value. This decline was exacerbated by network outages and technical issues, further eroding investor confidence and contributing to the token's downward trajectory for many months.

4 Priority Fees

4.1 Overview

In blockchain networks, transaction fees, often referred to as gas fees, are crucial for ensuring that transactions are processed in a timely manner. These fees incentivize validators or miners to prioritize certain transactions over others, especially during periods of high network activity these fees help prevent congestion by ensuring that users can secure their transaction landing by paying an extra based on the urgency. Without such a mechanism, the network could become clogged with low-priority transactions, destroying reliability for users and services that require a strict latency.

Ethereum, uses a gas fee model to manage its transaction processing. In this model, users pay gas fees to incentivize miners to include their transactions in the next block. The gas fee is determined by the complexity of the transaction and the current demand on the network. During periods of high congestion, gas fees can increase significantly, which helps prioritize transactions but can also make the network extremely expensive to use for smaller transactions.

Solana differentiates itself from Ethereum by offering a no-gas transaction model, meaning it aims to provide a high throughput with a base fixed transaction costs. However, Solana still introduced a form of priority fees to manage transaction processing during times of network congestion, this was primarily due to events of serious outages caused by nft spam in the past [22]. Transactions that include priority fees are placed higher in the queue, ensuring they are processed ahead of others. This system helps manage the network load by allowing users who need faster transaction confirmation to pay for that privilege, thus preventing congestion from delaying critical transactions. [23]

While Ethereum relies heavily on variable gas fees that fluctuate with network demand, Solana aims to maintain low transaction costs while still providing a mechanism for prioritization. This makes Solana potentially more accessible for everyday users who might be priced out during peak times on Ethereum. This also makes Solana's network more predictable and user-friendly, particularly for decentralized applications (DApps) that require consistent and affordable transaction processing.

Furthermore, as we are going to see later, prioritization depend highly on which locks the transaction requires, meaning that in periods of normal network activity a spike on a given account (for example an Nft drop) would have a minimal impact on the average fee for the whole network, while growing as necessary for that particular resource. [24]

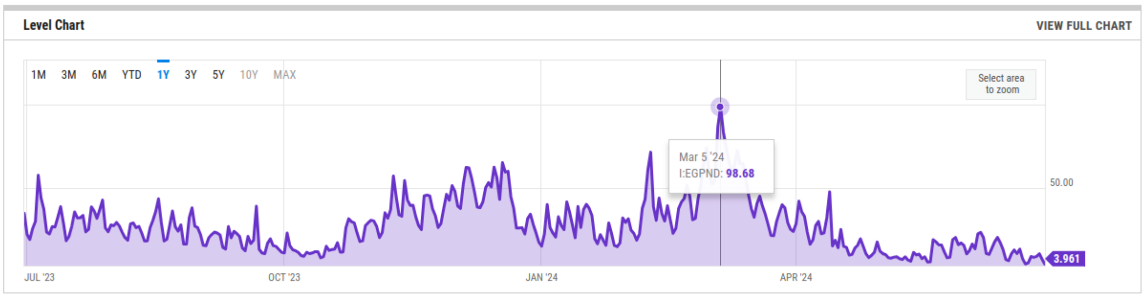


Figure 7: Example of extreme gas price fluctuations over last year

However, there are some drawbacks to Solana's approach. Despite its priority fee system, Solana can still experience unusually high demand, particularly when new applications or tokens suddenly increase traffic. This can lead to transaction delays, even for those who have paid for prioritization and in periods of extreme congestion the priority fees could end up creating a marked just like gas fees' one.

4.2 The failure

As we already described, during the April 2024 crisis, the Solana blockchain faced severe congestion, leading to significant transaction failures. Despite Solana's promise to handle high loads effectively, the network did not perform as expected under the intense pressure of increased activity driven by spam.

The community expected that priority fees would help mitigate this issue, this was the entire reason for priority fees existing and finally it was being tested. However, this expectation was not met. One of the main reason was actually very naive, most of the application interacting with solana did not support priority fees: since they had never been particularly necessary most services never implemented the option to pay an extra for their users transactions to secure a spot in the next block.

Another problem was also that, since transaction priority was rarely a problem and most services completely rely on the gas-free use of Solana, very little effort was put in requesting a number of ComputeUnits optimal for a given operation. Often transactions would contain extremely overshot CU limits, orders of magnitudes grater that the necessary ones. This made even more difficult correctly assessing the priority of incoming transactions.

This said, Solana users that instead were actually paying high priority fees with the correct CUs limit were expecting their transactions would be prioritized and included among the few that were approved. However, also this expectation was not met. Despite the additional fees, high-priority transactions were still frequently dropped or processed later than lower-fee transactions. This inconsistency highlighted a significant flaw in Solana's transaction scheduling and prioritization mechanisms.

Overall, the crisis exposed critical weaknesses in Solana's ability to manage transaction prioritization under heavy congestion. The inability to reliably process high-priority transactions despite additional fees eroded user trust and demonstrated the need for more robust solutions to handle such scenarios in the future.

4.3 The fix

Following the disaster, Solana quickly announced that the problem had been addressed in version 1.18, which was slated for release. However, upon examining the change logs for this version, there were no specific details about what had been altered to resolve the prioritization issues [25]. Investigations on social media platforms like Twitter and Reddit revealed that most discussions about the new release only briefly mentioned "fixed prioritization" without providing substantial details. [26]

A closer look into the commits for version 1.18 revealed a crucial clue in the form of a pull request titled "Scheduler - prioritization fees/cost #34888." This pull request indicated a minor fix in the calculation of the price per compute unit set by the user. The old computation did not adequately consider the base fees collected, leading to suboptimal resource allocation. The adjustment aimed to ensure that the compute unit price accurately reflected the user-defined priority fees, which is essential for the effective prioritization of transactions. [27]

$$Priority = \frac{\text{Priority Fee}\times\text{Compute Units Requested}+\text{Base Fee}}{1+\text{Requested Execution CUs}+\text{Signature CUs}+\text{Write Lock CUs}}$$

However, the impact of this change was minimal on the overall compute unit price, suggesting that this adjustment alone could not be the comprehensive solution to the problem. But starting from that clue and investigating into other changes from the same author and on those files the real improvement was finally discovered. A completely new transaction scheduler was added: the ‘CentralScheduler‘.

5 Scheduling

At the beginning of each epoch, the Solana cluster creates a leader schedule based on the stake-weight of each validator. This schedule allocates groups of four slots to validators in proportion to their stake-weight. During these allocated slots, the validator becomes the leader responsible for block production. The process of creating a block involves selecting and including transactions received from users and other validators in the network.

However, each block must be executed within 400ms, imposing limits on the amount of work a block can contain. Consequently, leaders often receive more transactions than can be included in a block, necessitating a decision-making process to determine which transactions to prioritize. Here comes the role of the scheduler, deciding the order in which to process incoming transactions.

To understand how the new CentralScheduler improves prioritization, we first need to analyze the old ThreadLocalMultiIterator scheduler. This section will explore the legacy scheduler, identify potential causes of prioritization inconsistencies, and systematically replicate the problem in a controlled environment. By examining the legacy scheduler, we aim to pinpoint why high-priority transactions were often delayed or dropped during periods of extreme network congestion.

Following the analysis of the old system, we will introduce the new CentralScheduler, which was implemented to address these issues. We will explain the design and functionality of the CentralScheduler, detailing how it enhances transaction prioritization and improves overall network performance. Finally, we will conduct empirical tests to verify that the new scheduler resolves the prioritization issues observed in the previous system.

The information in the following sections are sourced via analysis of the code [26] and declarations from the main contributor [28] [29] [30]

5.1 Legacy scheduler

The first solana scheduler

Solana's transaction history is composed of a series of entries, each containing a list of transactions. These entries cannot include conflicting transactions, as any conflict would render the entire block invalid. The block-production process, situated between signature verification (SigVerify) and broadcasting (BroadcastStage), historically relied on multiple nearly independent threads, each with its own queue of transactions and scheduling process.

In this historical design, each thread accessed a shared channel to pull transactions into its local queue, where transactions were sorted by priority. During block production, threads would attempt to grab locks on the top 128 transactions from their queue. If a lock grab failed due to conflicting transactions, the transaction would be retried later.

This design had significant drawbacks, especially during competitive events like NFT mints, where the top of the queue often contained conflicting transactions, leading to inefficiencies.

ThreadLocalMultiIterator

To address these inefficiencies, the multi-iterator approach was introduced, significantly refining the transaction scheduling process while retaining the architectural framework of historic designs. The multi-iterator manages multiple pointers (iterators) that simultaneously traverse a set of transactions, optimizing the selection process for inclusion in a block. This approach operates on a decision function that assesses each transaction based on criteria such as priority and potential for conflict, ensuring that only the most suitable transactions are processed together. It begins by arranging transactions into a serialized vector, providing a structured sequence from which to form batches. Each iterator is placed to start from different points in this vector, skipping transactions that conflict with those already selected in the current batch, thereby minimizing lock conflicts and retries, especially during high network activity such as large-scale token drops or NFT mints.

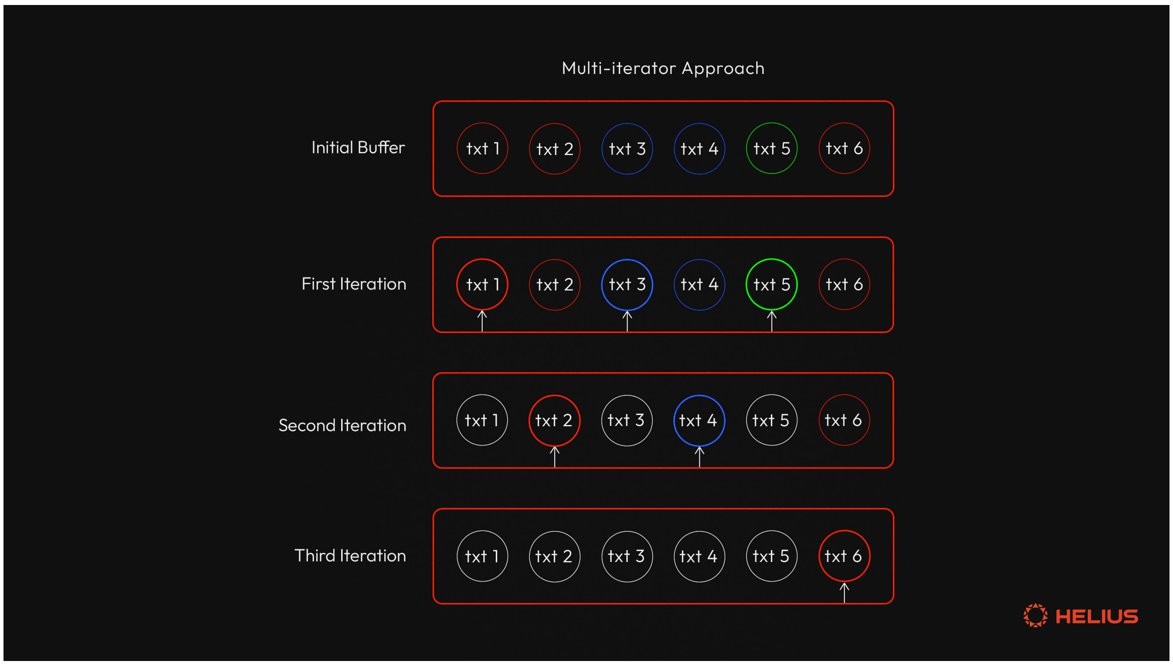


Figure 8: Multi iterator window depiction. Source: [30]

The decision function continuously evaluates whether a transaction should be processed immediately, deferred, or skipped based on its compatibility with the current batch. During block production, each BankingStage thread takes the top 128 transactions from its local queue and attempts to grab the necessary locks. If a lock grab fails due to a conflict, the transaction is retried in the next iteration, preventing bottlenecks. This retry mechanism is crucial for managing high transaction volumes, ensuring no thread is idle or overloaded.

The multi-iterator's check-lock mechanism preempts conflicts by verifying locks against the in-progress batch before selection, reducing retries and enhancing efficiency. It also allows lower-priority, non-conflicting transactions to be included in the block, optimizing resource utilization. The integration of the multi-iterator within BankingStage threads allows for independent yet coordinated processing, where each thread forms its own non-conflicting batches. The serialized vector serves as a shared resource from which threads pull transactions, but processing remains localized within each thread's scope, enhancing performance and scalability.

5.1.1 Local queue

Despite the efficiency improvements brought by the multi-iterator approach in Solana's BankingStage, the architecture still encounters significant challenges, particularly during high-traffic events. Each of the four threads dedicated to processing non-vote transactions operates independently, drawing transactions from a shared channel after they pass through the SigVerify stage. This randomness in transaction distribution means that each thread holds a unique subset of transactions, potentially including multiple high-priority conflicting transactions during peak times, like popular NFT mints. Consequently, even though the multi-iterator helps organize transactions within a single thread to avoid conflicts, the separate threads may still experience inter-thread locking conflicts. These arise because the threads, processing different sets of transactions deemed high-priority within their own queues, race against each other. This competition can lead to repeated and unsuccessful lock attempts, causing bottlenecks and a decrease in overall system efficiency. More importantly as we said each thread creates its own priority based on the transactions randomly assigned to it, this means that there is no mechanism avoiding many high priority transactions getting stuck enqueued in a thread while another one keeps processing relatively low priority one, causing the low priority ones to be processed first. This is the root cause of the problem we observed with prioritization inconsistencies.

5.1.2 Replicating the prioritization jitter

Our first goal is to replicate the described edge case to observe how low-priority transactions can be processed before higher-priority ones.

Testing Environment

We cloned the Solana repository and spawned a local test validator node. To increase visibility and properly understand what is going on, we made several changes to the validator code but replicated all results in a clean instance from the Solana CLI.

To test transaction order, we created a simple smart contract that maintains an internal state tracking received transactions. This contract has two exposed methods: one to add a transaction with a given ID and another to retrieve the stored transactions.

Listing 1: Partial testing program -Rust

```rust
#[program]
pub mod tx_order {
    use super::*;
    // Exported declarations
    pub fn record_transaction(ctx: Context<RecordTransaction>,
        transaction_id: u64) -> Result<()>

    pub fn get_transactions(ctx: Context<GetTransactions>)
        -> Result<Vec<TransactionRecord>>

    pub fn get_paged_transactions(ctx: Context<GetTransactions>, page: u64)
        -> Result<Vec<TransactionRecord>>

    pub fn get_transaction(ctx: Context<GetTransactions>, index: u64)
        -> Result<TransactionRecord>
}

#[account]
pub struct TransactionLog {
    pub records: Vec<TransactionRecord>,
}

#[derive(Clone, AnchorSerialize, AnchorDeserialize)]
pub struct TransactionRecord {
    pub account_owner: Pubkey, // 32 bytes
    pub transaction_id: u64, // 8 bytes
}

#[derive(Accounts)]
pub struct RecordTransaction<'info> {
    ...
}
...
```

For our test, we sent a series of transactions concurrently, to make the results more readable, each with a priority fee inversely proportional to the transaction ID, expecting transaction number 1 to be processed first and transaction number N to be processed last.

Our only focus was on transaction priority and not on conflicting transaction rescheduling, to factor out the conflicts overhead we could have either made all transactions independent or all conflicting, we made all transactions conflicting. This approach simplifies the test since storing all received transactions in a transactionsLog account provides a single point of conflict for all transactions.

To quantify prioritization inconsistencies, we tested various metrics, including:

- The module distance between the original ID and the resulting index of the transaction.
- The length of increasing sub-sequences.
- The number of inversions (number of times $TransactionLog[i]! = TransactionLog[i+1] - 1$).
- The number of inversions within a larger window (next two or three transactions in order).

Ultimately, the number of inversions proved to be the most descriptive metric for the problem. Each inversion indicates that a lower-priority transaction was processed before its time. The module difference, while a natural choice, was unusable for reasons described later.

As thoroughly explained, we expect to find inconsistencies due to the local queues that each thread creates. To establish a baseline and prove this theory, we performed the experiment with an increasing number of threads assigned to the validator, ranging from 1 (where we expect no inconsistencies) to 4 (the standard number of cores assigned). Actually the real number of threads assigned to the validator node is 3 to 6, however we don't consider the two cores dedicated to voting transactions, which operate differently.

One of the modifications we implemented in the validator was to introduce a sleep delay in the BankingStage's main loop. The reason for this was that each step in the BankingStage sends received packets to individual threads, where each thread processes the transactions according to its priority. To ensure our tests were consistent, we needed all transactions to be received by the BankingStage in a single step. Even when running locally and sending all transactions concurrently, achieving this was challenging, leading to unreliable results. Introducing a 100 ms sleep delay helped ensure that most transactions would be processed in the same slot. However, there were still some inconsistencies when transactions arrived precisely between one step and the next. This issue rendered the modulo difference a poor indicator of transaction order accuracy. In contrast, the number of inversions proved to be an ideal metric, as it does not depend on the number of transaction arrivals prior and after a step change but maintains a consistent baseline.

Example sequence in which the Modulo difference proves inadequate in measuring jitter in case of inter-step arrival:

$InputTransactions : [5, 6, 4, mainLoopStep, 3, 2, 1]$
$TransactionLog : [4, 5, 6, 1, 2, 3]$
$ModuloDifference : 18$
$NumberofInversions : 1$

Results
The following results were gathered by sending 200 transactions concurrently to either the Solana validator or our custom version, while varying the number of threads each time:

- **Custom validator:** Starting with one non-voting thread, we observed a baseline of 1 inversion Example sequence in which the Modulo difference proves inadequate in measuring jitter in case of inter-step arrival.due to inter-step transaction arrival, but aside from that, all transactions were processed in the correct priority order. This outcome was expected since a single thread equates to a single queue, which necessarily orders transactions correctly by priority.
  As we increased the number of threads, we noted a clear increase in the number of inversions, aligning with our expectations. Each additional thread represents another priority queue that could potentially conflict with others.

N of Threads	**3**	**4**	**5**	**6**
Median Inversions	1	3	6	8

Table 1: Median number of inversions in the slow validator with the old scheduler

- **Authentic validator:** Confirmed our theory, we then conducted the same experiment using an instance of the validator from the Solana CLI. As anticipated, the non-slowed-down version of the validator exhibited a significantly higher baseline of inconsistencies, even with a single thread, due to transactions arriving across multiple steps of the main loop.
  Nevertheless we observed a similar pattern of an allarming increase in inconsistencies as the number of threads increased.

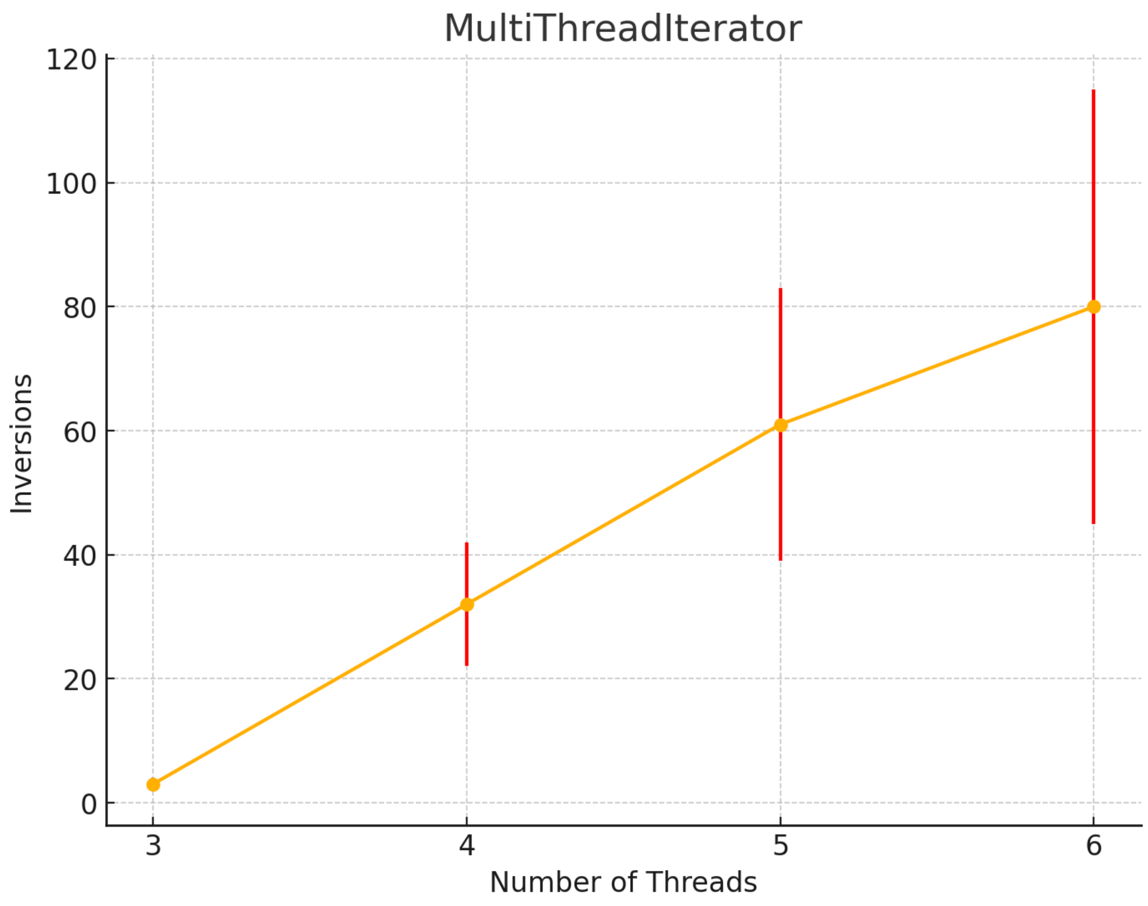


Figure 9: MultiThreadIterator scheduler inversions/threads dependency

These tests clearly demonstrated that under conditions of extreme network load, transaction priority is often not respected, leading to a significant number of inconsistencies. Moreover, this was a highly controlled environment with controlled conflicts, a single program, and all transactions requesting the same amount of compute units and performing the same calculations, therefore taking the same amount of time. It's easy to imagine how, when including variables such as different transaction types and computational demands, the likelihood of a high-priority transactions lagging becomes even more common.

5.2 Central scheduler

The introduction of the central scheduler in Solana's BankingStage marks a significant evolution in transaction processing, addressing key inefficiencies of the previous multi-iterator approach. This section provides a detailed technical description of how the new scheduler works and then we will empirically prove that the new scheduler fixed the prioritization issues.

Central Scheduling Architecture

The central scheduling architecture replaces the model of having multiple independent threads each managing their own transaction prioritization and processing. Instead, a single scheduling thread now manages transaction intake and distribution to worker threads. This centralized approach ensures consistency in priority handling and reduces the potential for inter-thread conflicts.

The architecture consists of a scheduler that is the sole thread accessing the channel from SigVerify. The scheduler pulls transactions from the receiver channel and sends them to the appropriate worker thread. Separate channels are used between the scheduler and worker threads, allowing conflicting work to be queued onto the thread it conflicts with, thus optimizing overall throughput. The scheduler maintains a view of which account locks are in use by which threads, enabling it to determine which transactions can be queued on which threads. Worker threads process batches of transactions in the received order and send a message back to the scheduler upon completion. These messages update the scheduler's view of the locks, informing the scheduling of future transactions.

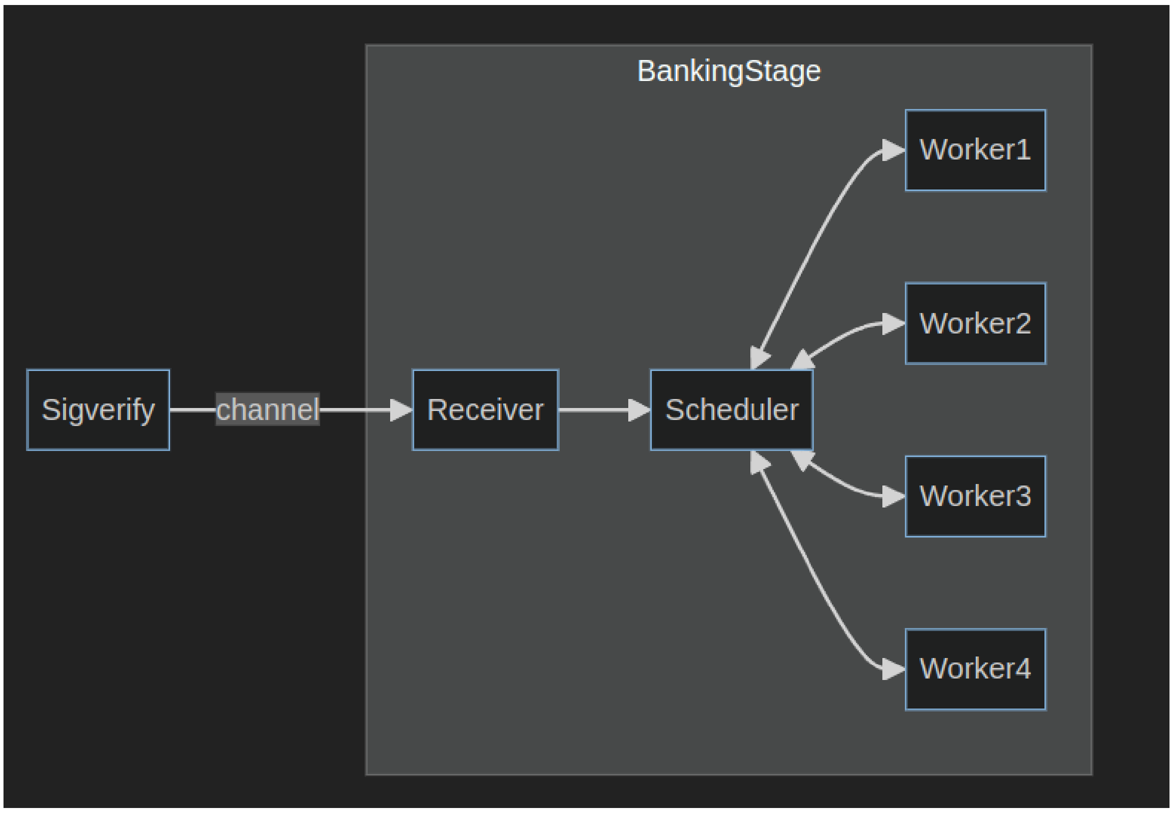


Figure 10: Transactions flow with central scheduler. Source: [28]

Scheduling Algorithm: Prio-Graph

The central scheduler uses a sophisticated algorithm involving a priority queue and a transaction dependency graph known as a prio-graph. This directed acyclic graph (DAG) is lazily evaluated as new transactions are added. The highest-priority transactions are popped from the queue and inserted into the prio-graph. Edges in the graph represent dependencies, where a transaction depends on a higher-priority transaction if they conflict on the same account.

The scheduling process begins by popping a "look-ahead" window of transactions from the priority queue and inserting them into the prio-graph. As transactions are processed, the scheduler pops from the prio-graph and continually adds new transactions from the priority queue. This allows the scheduler to look ahead and determine if currently non-conflicting transactions will eventually conflict with future transactions, enabling better thread assignment to avoid unschedulable states.

Benefits of the Central Scheduler

- **Reduction in Processing Delays:** The prio-graph helps ensure that transaction batches prepared for execution are highly likely to succeed without lock conflicts, streamlining processing time and reducing delays caused by lock contention.
- **Scalability and Flexibility:** The centralized view of locks and controlled transaction distribution among workers allow for an increase in the number of threads without the previous concerns of increased lock conflicts. This design is more scalable and flexible, accommodating higher transaction throughput.
- **Consistency in Priority Handling:** By centralizing the transaction intake and scheduling, the new system ensures that all transactions are processed in a consistent priority order. This eliminates the jitter caused by multiple threads having different views of transaction priorities.

5.2.1 Proving the prioritization consistency

Given the structure of the new central scheduler, we expect the jitter in transaction prioritization to be completely eliminated, though we still anticipate a baseline due to inter-step transaction receiving. Similar to the previous tests, the following results were gathered by sending 200 transactions concurrently to either the Solana validator or our custom version, while varying the number of threads each time:

- **Custom validator:** Starting with one non-voting thread, we observed the same baseline of 1 inversion due to inter-step transaction arrival, but aside from that, all transactions were processed in the correct priority order. This matches the results of the previous scheduler.
  As we increased the number of threads, the number of inconsistencies did not increase. Since we have a common scheduling thread, increasing the number of worker threads does not affect the order of processed transactions.

N of Threads	**3**	**4**	**5**	**6**
Median Inversions	1	1	1	1

Table 2: Median number of inversions in the slow validator with the new scheduler

- **Authentic validator:** We then conducted the same experiment using an instance of the validator from the Solana CLI. As expected, the baseline of inconsistencies was higher due to the larger number of transactions arriving across different steps of the main loop.
  However, this number did not depend on the number of threads, demonstrating that the central scheduler maintains consistent prioritization regardless of the number of worker threads.

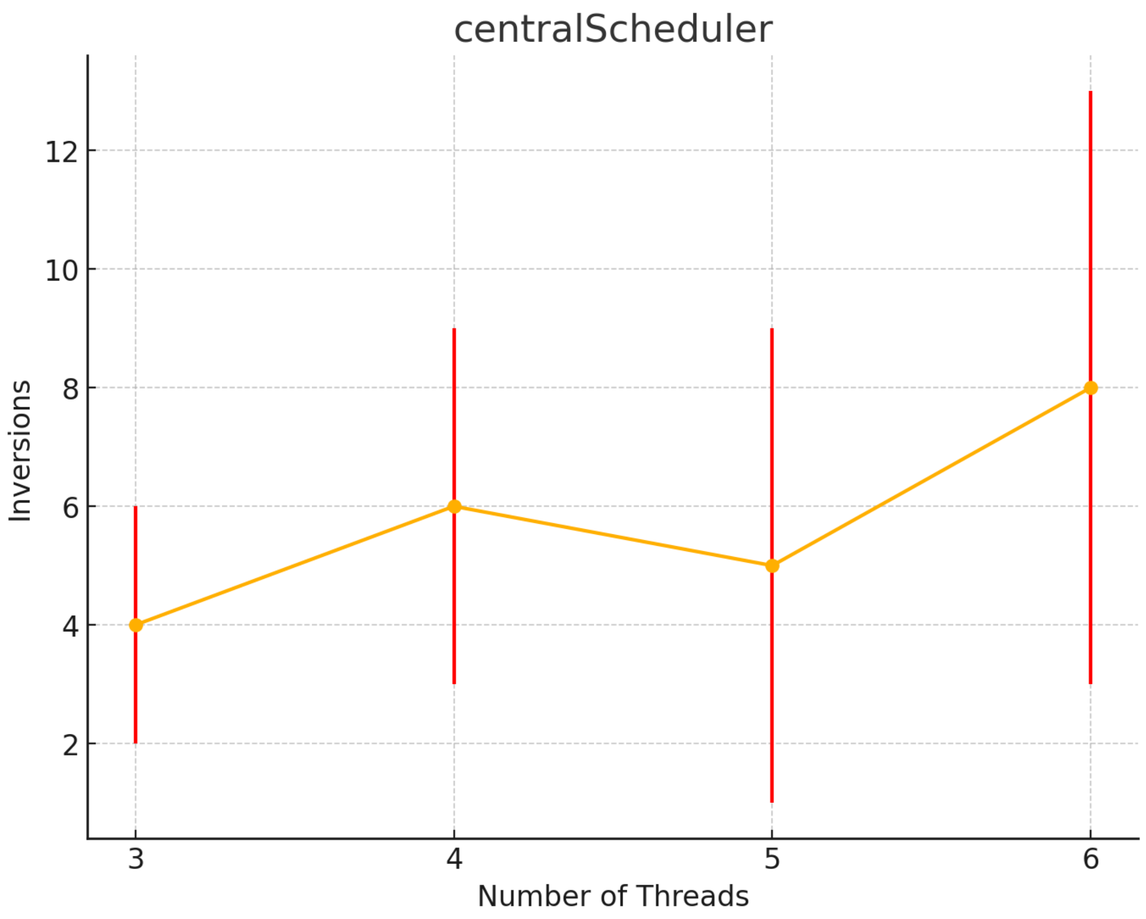


Figure 11: CentralScheduler inversions/threads dependency

Conclusion We have proven that the new scheduler, among other benefits, successfully addresses the prioritization inconsistency issues that led to user dissatisfaction during the April crisis. The new scheduler is a significant step towards making Solana the reliable and efficient network that was promised. Its implementation ensures consistent transaction prioritization, reduces processing delays, and enhances scalability, contributing to a more stable and performant blockchain platform.

6 Conclusions

In this thesis, we have provided a comprehensive analysis of the Solana blockchain, focusing on its development, the significant April 2024 congestion crisis, and the subsequent impact on stakeholder trust and network performance. We began by examining the rapid growth of Solana, driven by its innovative Proof of History consensus mechanism and advanced architectural features, which have enabled it to achieve high throughput and low transaction costs.

We examined how the April 2024 crisis exposed critical vulnerabilities in Solana's infrastructure, particularly highlighting the unreliability of the priority fees system designed to manage network congestion. This analysis revealed the necessity for effective priority fee mechanisms to ensure efficient transaction processing during periods of high demand. We explored the inner workings of these fees and detailed how and why they failed under the intense pressure of the surge in network activity, resulting in widespread transaction delays and failures.

Central to addressing these issues was the reengineering of Solana's scheduling mechanisms. We dived into the technical details of Solana's legacy schedulers and the introduction of the new CentralScheduler. Through detailed analysis and replication of the crisis conditions, we demonstrated how the new scheduler successfully addresses the prioritization inconsistency issues that led to user dissatisfaction. The new scheduler ensures consistent transaction prioritization, reduces processing delays, and enhances scalability, contributing to a more stable and performant blockchain platform.

While the implementation of the new scheduler is a significant step towards making Solana the reliable and efficient network it promises to be, the hit on stakeholder trust has been substantial. Moving forward, Solana will need to continue proving its capability to handle extreme network loads and maintain its high throughput. The ongoing development and enhancement of Solana's infrastructure will be crucial in restoring confidence and demonstrating its potential as a leading blockchain network.

By exploring these themes in depth, this thesis aims to contribute valuable insights into the management and scalability of blockchain networks. Ensuring their resilience and efficiency in the face of future challenges is essential for the continued growth and adoption of blockchain technology.

7 Appendix

Listing 2: Complete testing program -Rust

```
#[program]
pub mod tx_order {
    use super::*;
    pub fn record_transaction(ctx: Context<RecordTransaction>,
        transaction_id: u64) -> Result<()> {
        let transaction_log = &mut ctx.accounts.transaction_log;
        let entry = TransactionRecord {
            account_owner: *ctx.accounts.user.key,
            transaction_id,
        };
        transaction_log.records.push(entry);
        Ok(())
    }

    pub fn get_transactions(ctx: Context<GetTransactions>)
        -> Result<Vec<TransactionRecord>> {
        Ok(ctx.accounts.transaction_log.records.clone())
    }

    pub fn get_paged_transactions(ctx: Context<GetTransactions>, page: u64)
        -> Result<Vec<TransactionRecord>> {
        let records_per_page =25;
        let start = (page * records_per_page) as usize;
        let end = ((page + 1) * records_per_page) as usize;

        let records = &ctx.accounts.transaction_log.records;
        let page_records: Vec<_> = records[start..end].iter().cloned().
            ↪ collect();

        Ok(page_records)
    }

    pub fn get_transaction(ctx: Context<GetTransactions>, index: u64)
        -> Result<TransactionRecord> {
        Ok(ctx.accounts.transaction_log.records[index as usize].clone())
    }
}

#[account]
```

```
pub struct TransactionLog {
    pub records: Vec<TransactionRecord>,
}

#[derive(Clone, AnchorSerialize, AnchorDeserialize)]
pub struct TransactionRecord {
    pub account_owner: Pubkey, // 32 bytes
    pub transaction_id: u64, // 8 bytes
}

#[derive(Accounts)]
pub struct RecordTransaction<'info> {
    #[account(init_if_needed, payer = user, space = 10_000)]//mut // ~ 200
        ↪ transactions
    pub transaction_log: Account<'info, TransactionLog>,
    #[account(mut)]
    pub user: Signer<'info>,
    pub system_program: Program<'info, System>,
}

#[derive(Accounts)]
pub struct GetTransactions<'info> {
    pub transaction_log: Account<'info, TransactionLog>,
}
```

References

[1] Coindesk. Solana accepts the challenge. https://www.coindesk.com/tech/2024/03/20/solanas-yakovenko-welcomes-meme-coin-traders-with-nothing-better-to-do/.

[2] Anatoly Yakovenko. Solana cofounder twitter account. https://x.com/aeyakovenko.

[3] CNN. Solana congestion crisis. https://www.ccn.com/news/crypto/solana-congestion-crisis-what-solution.

[4] Blockworks. Solana congestion. https://blockworks.co/news/solana-network-congestion-transaction-volume.

[5] Elliots Wave. Crisis impact. https://elliottswave.com/2024/05/08/a-spring-downfall-solana-ecosystem-and-sol-price-action-in-april-2024.

[6] Helius. Solana quic implementation. https://www.helius.dev/blog/all-you-need-to-know-about-solana-and-quic.

[7] Anza (ex Solana Labs). Solution timeline post. https://x.com/anza_xyz/status/1776347491195388156.

[8] Coinlive. Known bottleneck. https://www.coinlive.com/news-flash/488829.

[9] Austin Federa. Atrategic director of the solana twitter. https://x.com/Austin_Federa.

[10] WuBlockchain. Post on quic problem. https://x.com/WuBlockchain/status/1776452218063049024.

[11] Coinedition. What happened to solana transactions. https://coinedition.com/solanas-team-lead-ensures-known-solutions-for-transaction-failures/.

[12] Dailycoin. Why are transactions failing. https://dailycoin.com/why-are-solana-transactions-failing-devs-scramble-to-fix-it.

[13] Medium. History of solana. https://medium.com/@tempestgirl1/the-history-of-solana-2ffd1f9b560e.

[14] Dailycoin. History of solana, the evm killer. https://dailycoin.com/solana-history-story-behind-cryptos-leading-ethereum-killer/.

[15] Forbes. What is solana. https://www.forbes.com/advisor/investing/cryptocurrency/what-is-solana/.

[16] Solana. Uptime report march 2024. https://solana.com/news/network-performance-report-march-2024.

[17] JumpCrypto. Firedancer, a new validator. https://jumpcrypto.com/writing/firedancer-reliability/.

[18] Alchemy. Firedancer. https://www.alchemy.com/overviews/what-is-firedancer.

[19] blockworks. Feb 2024 outage. https://blockworks.co/news/solana-downtime-post-mortem.

[20] Solana. Feb 2024 outage postmortem. https://solana.com/news/02-06-24-solana-mainnet-beta-outage-report.

[21] Messari.io. 2024 solana report. https://messari.io/project/solana/quarterly-reports/q1-2024.

[22] Coindesk. Priority fees introduction. https://www.coindesk.com/tech/2023/02/24/for-solana-users-priority-fees-mean-paying-up-to-skip-the-line/.

[23] Helius. Priority fees. https://www.helius.dev/blog/solana-fees-in-theory-and-practice.

[24] Solana. How to use priority fees. https://solana.com/developers/guides/advanced/how-to-use-priority-fees.

[25] Solana. Blockoptimization (crisis fix). https://solana.com/news/block-optimization-on-the-solana-network.

[26] Solana Labs. v1.18 commit logs. https://github.com/anza-xyz/agave/wiki/v1.18-commits.

[27] Andrew Fitzgerald. Priority calculation pr. https://github.com/solana-labs/solana/pull/34888.

[28] Andrew Fitzgerald. Solana banking stage and scheduler. https://apfitzge.github.io/posts/solana-scheduler.

[29] Solana. Andrew fitzgerald interview on the solana scheduler. https://www.youtube.com/watch?v=R7hq8ampBio&ab_channel=Solana.

[30] Helius. All you need to knwo about 1.18. https://www.helius.dev/blog/all-you-need-to-know-about-solanas-v1-18-update.