# Turbine: Block Propagation on Solana

## What's this Article About?

Data availability is crucial to blockchains. It ensures that all necessary information is readily accessible to nodes for validation purposes, which maintains the integrity and security of the network. However, ensuring data availability while maintaining a high level of performance is a significant challenge, especially as networks scale.

Solana has risen to this challenge with a unique architectural design that facilitates continuous block creation and propagation. This is made possible through several key innovations such as leader selection, Gulf Stream (eliminates the need for a mempool), and Turbine (block propagation mechanism).

The continuous nature of Solana necessitates an efficient system to ensure that all validators receive the most updated state promptly. A simplistic approach would have the leader transmit all blocks directly to every other validator. Yet, given Solana's high throughput, this method would significantly bloat bandwidth and other resource requirements while undermining decentralization.

Bandwidth is a scarce resource, and Turbine is Solana's ingenious solution to optimize information propagation from the leader of a given block to the rest of the network. Turbine was specifically designed to mitigate the stress of egress (sending data out) from the leader to the network

.

In this article, we'll dive into how Turbine works as well as its pivotal role in Solana's broader landscape of transaction inclusion. We'll also compare Turbine to other data availability solutions and discuss open research avenues in this domain.

## What's Turbine?

Turbine is a multi-layer block propagation mechanism used by aSolana cluster to broadcast ledger entries to all nodes. The core ideas behind Turbine have been in the mind of academics for many years as evidenced by this paper published in 2004 as well as more recent work.

Unlike traditional blockchains where a block is sent to all nodes sequentially or in a flooded manner, Turbine takes a more structured approach to minimize communication overhead and reduce the load on individual nodes. At a high-level, Turbine breaks a block down into smaller pieces and disseminates these pieces through a hierarchy of nodes. Here, an individual node does not need to be in contact with every other node, and only has to communicate with a select few. This becomes increasingly important as the network grows in size as traditional propagation methods would become untenable due to the sheer volume of necessary communications. Thus, Turbine ensures the fast and efficient dissemination of data across Solana. The speed at which blocks are propagated and verified is crucial to maintaining Solana's high throughput and network security.

Moreover, Turbine addresses the issue of data availability, ensuring that all nodes can access the required data to validate transactions in an efficient manner. This is done without requiring an enormous amount of bandwidth, which is a common bottleneck in other blockchain networks.

Turbine significantly contributes to Solana's ability to handle high transaction volumes and maintain a lean and efficient network structure by alleviating bandwidth bottleneck and ensuring rapid block propagation. This innovative protocol is one of the cornerstones that enables Solana to deliver on its promise as a fast, secure, and scalable network.

Now, let's delve deeper into the mechanics of Turbine and how it propagates blocks across the Solana network.

## How Does Turbine Propagate Blocks?

Before a block is propagated (i.e., transmitted to other validators in the network), the leader builds and orders the block given the incoming stream of transactions. After the block is built, it is ready to be sent via Turbine to the rest of the network. This process is referred to as block propagation.

Vote messages are then passed between validators, and these messages are encapsulated within the block data to satisfy either the commitment status "confirmed" or "finalized". A confirmed block is a block that has received a super majority of ledger votes whereas a finalized block is a block that has been confirmed and has 31+ confirmed blocks built atop of the target block. The difference in commitment status is explained more here. This part of consensus will be explored in a future post.

While leaders build and propose entire blocks, the actual data is sent as shreds

(partial blocks) to other validators in the network. Shreds are the atomic units sent between validators.

At a high level, Turbine takes shreds and sends them to a predetermined set of validators, who then relay those shreds to a

new set of validators. The following diagram outlines the continuous process of shred propagation:

In this example, Validator 1 is the designated slot leader. During its slot (validators are assigned leader for 4 consecutive slots), Validator 1 builds and proposes a block. Validator 1 first breaks the block into sub-blocks called shreds

through a process called shredding

. Shredding splits the block data into Maximum Transmission Units (MTU)-sized data shreds

(the maximum amount of data that can be sent from one node to the next without fragmenting it into smaller units) and generates corresponding recovery shreds

via the Reed-Solomon erasure coding scheme. This scheme aids in data recovery and ensures data integrity during transmission, which is crucial for maintaining the security and reliability of the network.

This process of shredding and propagation ensures a swift and efficient distribution of block data across Solana, maintaining high throughput and network security.

## Erasure Coding

Before shreds are propagated through the Turbine Tree, they are first encoded using Reed-Solomon erasure coding, a polynomial-based error detection and correction scheme. Erasure coding is used as a data protection method so that the original data can be recovered even if some parts are lost or corrupted during transmission. Reed-Solomon erasure coding is a specific type of Forward Error Correction (FEC) algorithm.

Because Turbine fundamentally relies on a series of packet retransmissions by downstream validators, those validators could either be malicious (adversarial Byzantine nodes) by choosing to rebroadcast incorrect data or receive incomplete data (network packet loss). Due to the retransmission tree structure of Turbine, any network-wide packet loss is compounded, and the probability of the packet failing to reach its destination increases on each hop.

At a high level, if the leader transmits 33% of the block's packets as erasure codes, then the network can drop any 33% of the packets without losing the block. Leaders have the ability to dynamically adjust this number (FEC rate) based on network conditions by taking into account variables such as recently observed network-wide packet loss and tree depth.

For simplicity, let's examine a shred group with an FEC rate of 4:4.

The data shreds

are partial blocks from the original block built by the leader while the recovery shreds

are the Reed-Solomon generated erasure coded blocks.

Blocks on Solana typically leverage an FEC of 32:32 (32 out of 64 packets can be lost with no need to re-transmit). As described by the Solana docs, these are the following conservative network assumptions:

- Packet loss rate of 15%
- 50k TPS generating 6400 shreds per second

An FEC rate of 32:32 yields a block success rate of ~99%. Furthermore, leaders have the agency to increase the FEC rate should they so choose to increase the probability of block success.

Turbine currently utilizes UDP for block propagation, providing immense latency benefits. According to a validator operator, transmitting 6 MB + erasure coding worth of data from us-east-1

to eu-north-1

using UDP takes 100ms while TCP takes 900ms.

## Turbine Tree

A Turbine Tree is a structured network topology used by Solana to facilitate the efficient propagation of shreds (encoded block data) among validators. Once shreds are properly encoded into their respective shred groups, they are ready to be disseminated through the Turbine Tree to inform other validators in the network of the most updated state.

Each shred group is sent via a network packet to a special root node which manages which validators are part of the first layer (1 hop away). The following steps are then executed:

1. List Creation:

The root node aggregates all active validators into a list, which is then sorted based on the stake each validator has in the network. Validators with higher stake-weight are prioritized to receive shreds sooner, enabling them to respond with their

own vote messages for consensus faster.

1. List Shuffling:

This list is then shuffled in a deterministic manner. This creates a “Turbine Tree” generated from the set of validator nodes for each shred using a seed derived from the slot leader id, slot, shred index, and shred type. A new tree is generated at run-time for each shred group to mitigate potential security risks associated with a static tree structure.

1. Layer Formation:

Nodes are then divided into layers, starting from the top of the list. The division is based on the DATA_PLANE_FANOUT

value, which determines the breadth and depth of the Turbine Tree. This value impacts how quickly shreds can be propagated through the network. The current DATA_PLANE_FANOUT is 200, so most validators are only 2-3 hops away (leader -> root -> L1 -> L2).

The Turbine Tree, being known to all, ensures each validator knows exactly where they are responsible for relaying that shred. The Turbine Tree is typically a 2- or 3-hop tree (depending on the number of active validators), given the current DATA_PLANE_FANOUT

value of 200.

Furthermore, nodes are able to fall back to gossip and repair if they don't get enough shreds or if the loss rate exceeds the FEC rate. Under the current implementation, a node lacking enough shreds to reconstruct the block sends a request to the leader for re-transmission. Under deterministic Turbine, any node that received the full block can send repair shreds that the requesting node needs, thus pushing data transmission further down to the areas of the tree that are requesting data.

## Comparing Block Propagation Between Solana and Ethereum

Block propagation on Solana is different than Ethereum. Here are some high-level differences:

- Solana’s ideal bandwidth requirements (>1 Gbps) are significantly higher than Ethereum’s (geth recommends >25 Mbps). This higher bandwidth requirement is attributed to Solana’s larger block sizes and quicker block times. Solana’s design allows for effective utilization of the entire bandwidth to expedite data transmission, thus reducing latency. While there are bandwidth spikes up to 1 Gbps, it is not consistently using 1 Gbps. Solana’s architecture specifically enables spikes in bandwidth demand.

- Solana employs Turbine for block data propagation whereas Ethereum utilizes a standard gossip protocol. On Ethereum, block data propagation is carried out in a straightforward manner: each node communicates with every other full node in the network. Once there is a new block, clients will verify it by sending it to their peers and approving the transactions that are in the block. This mechanism suits Ethereum due to its smaller block sizes and longer block times in comparison to Solana. When it comes to Ethereum L2 rollup data (excluding validiums), the propagation follows the gossip protocol as well, with block data stored in the “calldata” field of Ethereum L1 blocks.

- Ethereum uses the TCP (via the DevP2P protocol) for block propagation while Solana uses UDP (with some community support to transition to QUIC). There are some tradeoffs to be considered between UDP and QUIC:

- UDP’s uni-directionality leads to lower latency compared to QUIC, which necessitates QUIC streams. Discussions are ongoing regarding the implementation of uni-directional streams into QUIC.

- Advocates for QUIC assert that while custom control flow can be done over UDP, it demands a substantial engineering effort which QUIC alleviates by natively supporting such features. The end goal is the same but the upper bound of QUIC performance (latency, throughput, etc.) is the current state of pure UDP.

These distinctions underline the unique architectural decisions adopted by Solana and Ethereum, which contribute to their individual performance, scalability, and network robustness. For a more in-depth analysis on TCP, UDP, and QUIC, check out our article on Solana and QUIC.

## Future Research Questions

Block propagation and data availability remain as open research areas with numerous teams crafting their own unique approaches. While the metrics might evolve, we want to provide an overview of the different approaches and their associated tradeoffs:

- Some discussions have surfaced around Turbine’s standing as a “data availability” (DA) mechanism. Turbine serves as a data availability mechanism in the sense that the entire block data is being published and downloaded by all other validators on Solana. Nonetheless, Turbine lacks support for data availability sampling (DAS), a feature that aids light nodes in the verification of state with reduced hardware requirements. This is an active development focus for teams such as Celestia. Like Turbine, DAS also utilizes erasure codes but does so with the express purpose of detecting and preventing data withholding attacks.

- For Solana Virtual Machine (SVM) L2s such as Eclipse, Turbine loses relevance as there is no validator set to pass data between. In the case of Eclipse, block data is published to Celestia for data availability– this enables outside observers to run fraud proofs to ensure correct execution and state transitions. Eclipse will be one of the first implementations of the SVM outside of the Solana network itself. Pyth has also forked the SVM for its own oracle network called “Pythnet” and effectively runs as its own sidechain.
- On Solana, full nodes manage block propagation while also engaging in other segments of the integrated blockchain stack like transaction ordering and consensus. What would be the quantitative metrics for Turbine if operated as a modular component on specialized hardware?
- Turbine prioritizes higher stake-weight nodes to receive block data first. Will this lead to more MEV centralization over time?
- How will the different approaches to data availability such as EigenDA (horizontally-scalable single unicast relayer) and Celestia (data availability sampling) compare to Turbine in production with respect to raw throughput and trust minimization?
- Firedancer aims to further increase data propagation and is optimized for a robust 10 Gbps bandwidth connection. How will the system-level optimizations they have made around Turbine perform in production for consumer-grade hardware as well as professional-grade hardware?
- Currently, all nodes on Solana are full nodes (light client implementations are still under development). Sreeram Kannan (EigenLayer) recently described an implementation of DAS-S on top of Turbine. Will there be support for a version of DAS for Turbine? Can light clients with DAS be implemented to keep high data throughput while light clients (with much lower resource requirements) satisfy trust minimization?

## Conclusion

Congratulations! In this article, we examined Turbine and how it works within Solana’s broader landscape of transaction inclusion. We compared Turbine to other data availability solutions and discussed the different research avenues that are open to this domain. Solana’s Turbine protocol stands as a testament to the network’s commitment to achieving high throughput and low latency by leveraging a structured network topology to efficiently disseminate block data among validators.

Finding ways to enhance data availability and make block propagation more efficient drives innovation within the broader blockchain community. Comparative analysis of Solana and Ethereum’s block propagation mechanisms sheds light on their individual strengths and tradeoffs, as well as inspires a deeper conversation on how emerging blockchain solutions, such as EigenDA, Celestia, and Firedancer might shape this ecosystem in the future.

The solution to efficient data propagation and data availability is far from complete. However, Solana’s approach and steadfast commitment to optimizing network performance without compromising security and trust minimization is warmly welcomed.

Thanks to @dubbel06 and @jon_charb for review and comments.

# Additional Resources / Further Reading

- Turbine Block Propagation
- Understanding Erasure Coding
- Ethereum Blocks
- Pyth Perseus Network Upgrade
- Intro to EigenDA: Hyperscale Data Availability for Rollups
- Firedancer