PUSH CHAIN - Build Universal Apps

Any Chain. Any User. Any App.

Harsh Rajat
harsh@push.org

Richa Joshi
richa@push.org

Zaryab Afser
zaryab@push.org

Igor Artemev
igor@push.org

Aman Gupta
aman@push.org

Fabio Costa
fabio@push.org

Everett Muzzy
everett@push.org

Abstract

The paper introduces Push Chain, a Proof of Stake shared state blockchain for shared app experiences designed to support the development of hyper-scalable, consumer-centric, universal web3 applications.

The paper highlights how today's existing L1 and L2 chains are primarily optimized to support financial applications instead of consumer-centric applications. Additionally, the proliferation of Ethereum's rollup-centric roadmap has led to a fragmented ecosystem, complicating user experience and hindering web3's mass adoption.

Push Chain is built to address the current ecosystem's shortcomings by supporting scalable consumer apps with enhanced UX, thus driving the wave of app adoption.

Push Chain aims to resolve these issues with today's web3 app ecosystem in three phases. Phase 1 focuses on creating a decentralized network tailored for consumer applications, featuring innovations like any chain transaction, consumer transaction (order not important), fees abstraction (enabling fees from a chain of user choice or from web3 app), wallet abstraction, true scale with parallel block execution and dynamic sharding.

Phase 2 introduces seamless interoperability, arbitrary message bridging and cross-chain communication, while Phase 3 envisions a shared app experience, having universal smart contracts capable of interacting with any EVM or non-EVM chain, offering a seamless, unified user experience.

This paper primarily details the technical implementation of Phase 1.

Whitepaper Version 1.0 | August 2024 | Last Updated: 10th Dec, 2024

Index

1. The State of Web3 Applications

1.1 Web3 is not well-equipped for building consumer-centric apps

Web3, while revolutionary in its potential, is currently skewed heavily towards financial applications, particularly decentralized finance. This focus on DeFi is evident in the proliferation of blockchain projects that primarily revolve around trading, lending, staking, yield farming, etc. While these innovations have undeniably pushed the boundaries of what is possible with blockchain technology, they have also inadvertently narrowed the scope of web3's potential, overshadowing non-monetary consumer-centric applications that could drive mass adoption — like all the non-monetary apps we use today.

According to DeFi Llama, the total value locked (TVL) in DeFi protocols exceeded $100 billion in June 2024, showcasing the immense financial resources being poured into this sector. Popular chains like Ethereum, Binance Smart Chain (BSC), and Avalanche are dominated by DeFi projects. Moreover, the influx of new layer 1 and layer 2 chains has often been driven by the need to accommodate the high transaction volumes associated with DeFi. Chains like Solana and Avalanche, which boast high throughput and low transaction fees, have primarily attracted projects focused on decentralized exchanges (DEXs), lending protocols, and synthetic assets.

Despite the success of DeFi, there is a notable lack of consumer-facing applications that cater to the average user. The focus on financial instruments, which often require a deep understanding of blockchain technology and finance, has created a barrier to entry for non-technical users. Consumer-centric applications that could appeal to a broader audience, are still in their infancy compared to the well-established DeFi sector.

This focus on finance has led to a situation where the average user might view web3 as a playground for traders and investors rather than a tool for everyday life. It's now quite evident that for mass adoption to occur, web3 needs to expand beyond its financial roots and embrace a wider range of consumer applications that are appealing as well as provide benefits to the average user. These are users who are not DeFi experts or are solely driven by financial incentives to use an app. Instead, they require simpler apps that provide value in their daily lives without the extreme UX complexity that web3 apps currently have.

This is evidently also highlighted by Vitalik in his latest Devcon talk emphasizing the need for building non-financial and mixed financial apps.

However, web3 doesn't seem well-equipped for such apps currently, which is a major concern.

Mentioned below are just a few good reasons why it's extremely difficult to build consumer-centric applications in the current web3 ecosystem.

Requirements of a consumer-centric app vs. web3's current capabilities

Some of the core requirements of a consumer-centric application are:

1. **Simplified UX:** Consumer-centric applications require a seamless and intuitive user interface. This means being able to interact with an app without ever having to understand the complicated underlying chains, smart contracts, or native tokens.

   The current web3 ecosystem, however, has a lot of UX challenges starting from onboarding users to user interaction with the app by paying gas. Unlike traditional web apps, web3 requires users to grasp a variety of new concepts, which hinders mass adoption.

2. **Instant Finality & Scalability:** A consumer-centric app needs to support a large number of users without compromising performance. Additionally, it needs to support instant finality so that users experience a similar speed as in traditional apps.

   Although layer 2 solutions and rollups aim to address scalability, the current state of web3 still struggles to match the speed and efficiency of traditional centralized systems. For a consumer-centric app, any delay in processing or finalizing transactions will lead to a poor user experience.

3. **Affordability:** Consumers expect low or no fees for transactions within apps, especially non-monetary ones.

   Gas fees, while necessary to secure the network, are a major barrier for consumer adoption. In web3, transaction fees are often too high for the average consumer;

even more so during network congestion.

4. **Cost-Effective Storage:** For consumer apps to succeed, storage needs to be not only secure and decentralized but also affordable and scalable. Users of consumer apps expect to store and access data seamlessly, without worrying about costs or the complexity of interacting with decentralized networks.

   On the other hand, traditional blockchains currently have prohibitively expensive costs which act as a barrier for such apps.

5. **Shared App Experiences:** needs to be built which abstracts wallet, chain and fees. ie: any chain can send a transaction, wallet login is built for web3 users and for normies and fees can be paid either from the app, from a fees contract in native token of that respective chain, or via delegation of wallet.

For consumer-centric applications, which demand high performance and seamless user experiences, these features are absolutely required as without them, slow transaction times, high fees, complexity of securing transactions and apps bound to a specific blockchain ecosystem deter users from engaging with these platforms, further hindering the potential for mass adoption in the consumer space.

1.2 The rise of the multi-chain ecosystem and the fragmentation of web3

Since the advent of layer 1 chains, scalability and high fee (gas) requirements have been a major concern. For chains like Ethereum specifically, as the number of applications grew, so did the network congestion and overall cost for users.

A significant step to resolve these issues was taken on October 02, 2020, when Vitalik introduced the *Rollup-Centric Ethereum Roadmap*. This meant Ethereum moved towards rollups as their scaling strategy because they significantly increased transaction throughput as well as reduced transaction costs by bundling multiple transactions into batches.

This quite effectively solved the scalability issue for Ethereum effectively as it reduced load on the Layer 1 chain and provided cheaper transaction fees. However, we soon realized that the rollup-centric roadmap had led to the development of a plethora of new L2s, Rollup solutions, and L3s.

Additionally, the web3 ecosystem soon realized the need for designing chains that are specifically optimized for single applications, i.e. app-specific chains. This was necessary to ensure that these applications could achieve optimal performance, security, and user experience tailored to their unique requirements without being affected by the general-purpose network congestion and fees of more generic Layer 1 or Layer 2 solutions.

While app-specific chains achieved the desired outcome by optimizing for their specific needs, this again led to the creation of multiple new chains. L3s as applications like dYdX, Hyperliquid or Frax became standalone app-chains or rollups of their own.

This inception of new rollups, app-chains, L2 and L3s undoubtedly resolved quite a few key issues of the ecosystem like scalability, block space, network usage fees, etc. However, it also led to another significant concerns for users.

With the rise of all of these new chains, the overall UX of web3 has become increasingly complex for an average user to navigate. Users now need to manage multiple wallets, each possibly holding different tokens on various chains. Furthermore, the process of bridging assets between these chains is quite often cumbersome and fraught with risks. Users must rely on various bridging protocols, which not only add to the complexity but also introduce potential security vulnerabilities. For instance, a user willing to move tokens from Ethereum to an L2 solution must navigate multiple steps, including wrapping and unwrapping tokens, which can be a deterrent for new users.

This leads to an extremely complicated experience for end users where they not only have to keep track of multiple networks and assets but also deal with the risk of bridging assets between different chains.

The lack of a unified user interface results in users continually switching between different web3 apps and interfaces to manage their assets, track transactions, and engage with various protocols. This fragmented experience when people try to interact with web3 apps becomes a massive hurdle to the seamless experience that users need to facilitate mass adoption. The end result is that people will continue to only adopt DeFi apps at scale; putting up with frustration, risk, and poor UX is only worth it with the possibility of financial gain on the other side.

Moreover, with the rise of multi-chain ecosystem we have also started to witness the challenge of liquidity fragmentation across the many chains that exist. Decentralized exchanges (DEXs) and liquidity pools on each chain are isolated, meaning that liquidity is

not shared across the ecosystem. This isolation can lead to reduced liquidity on each platform, resulting in higher slippage and less efficient markets.

In summary, while the proliferation of L2s, rollups, and app-specific chains has advanced the scalability and functionality of the web3 ecosystem, it has also introduced significant UX challenges. To achieve mainstream adoption, it's now imperative that web3 finds ways to streamline these experiences, making it easier for average users to navigate the complex landscape of multiple chains and protocols.

2. Push Vision and Mission

Push's vision is a world of universal applications.

In this world, web3 applications look and feel like today's web2 counterparts, while retaining the core values and unique opportunities of blockchain-based protocols: censorship resistance, permissionlessness, resilience, and security. This app ecosystem is capable of handling global volumes of daily use, both financial and non-financial / mixed financial, without burdening everyday users with tedious choices between networks, wallets, assets, and more.

To date, web3 has created an extremely capable ecosystem of L1s, L2s, and (increasingly) L3s. These developments have solved for the core limitations of blockchain technology — largely issues of throughput, finality, sustainability, and privacy. Each of these protocols is vying to attract interest for and grow an app ecosystem that will bring in everyday users and keep them involved. Interacting with applications today, however, requires users to flip between blockchain protocols. This could mean setting up a new wallet, buying new assets, and/or bridging new assets — all of which takes time, is laden with risk, and hinders users from *just using the application*. To date, all this friction has not been a huge detractor from web3 adoption, as most of the industry's applications have been financial. People have and will put up with less-than-ideal UX for the possibility of financial gain. Now, however, we are on the cusp of a non-monetary consumer application boom, and the architecture of today's L1s, L2s, and L3s alone cannot support the type of experiences needed for those apps.

Today's protocol ecosystem can handle high app throughput, but it has still not achieved **true scale**. By true scale, we mean scale not just at the infrastructure level, but on the user experience level as well. As long as a user must select between specific blockchain protocols before using an application, web3 has not achieved true scale. True scale requires ease of use and allows quick decision making. It requires users to swiftly test out new apps and projects without burdening them with infrastructure-level choices. Applications must feel universal, not blockchain-specific.

Push's mission is to enable this world of universal applications through Push Chain.

Push Chain encompasses all of these needs and provides the infrastructure required to achieve true scale by enabling the creation of universal applications. Push Chain caters to all applications, regardless if they are monetary or non-monetary, and regardless of which

chain the user interacts from. In other words, Push Chain's mission is to enable everyone to just use an application, not use an application on X vs. Y vs. Z chain.

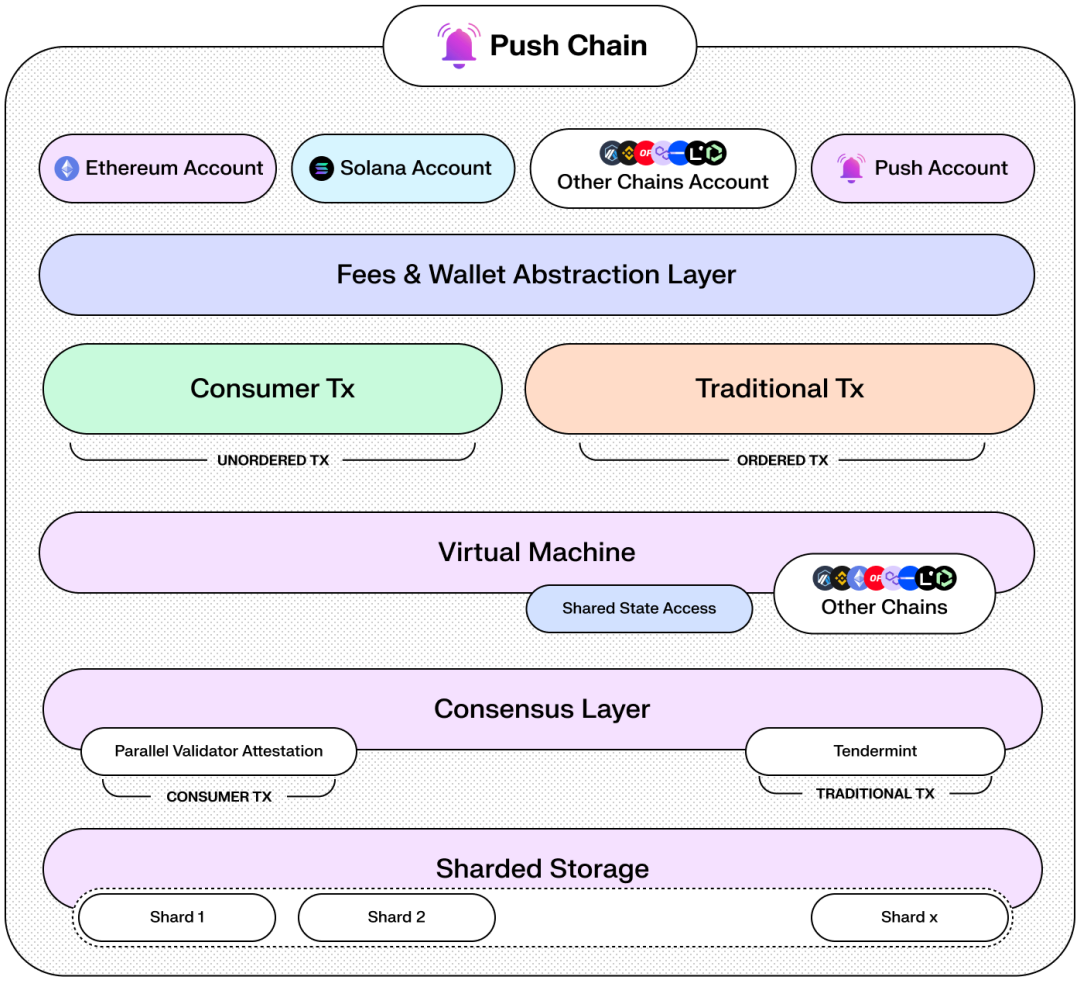
To create this future, Push Chain will launch in three phases: **1) consumer centric apps, 2) seamless interoperability, and 3) shared state and universal smart contracts.**

3. Introduction to Push Chain

Push Chain is a s*hared state blockchain* for *shared app experiences*. It is a Proof of Stake (PoS) chain built to allow developers to create Universal Apps and act as a shared settlement layer for multiple L1s, L2s and L3s. It allows transaction from any chain, uses gas abstraction to ensure no hassle for interaction from other chains and provides wallet abstraction for seamless onboarding and signing for existing web3 users as well as web2 normies.

It also introduces a new transaction type which we define as consumer transactions (where ordering is not important) to enable non-financial apps to have the speed and scalability which any consumer app requires. It leverages new mechanisms like parallel block execution and dynamic sharding to enable true scaling, providing the perfect environment for any consumer app or universal app to thrive in the era of fragmented chains.

Additionally, Push Chain is situated to form the **infrastructure for onchain AI.** Push Chain's shared state, fast finality, sharding, tx payload size, and ability to have users from any chain allow it to support fast, multi-use AI use cases (agents, apps) across all of web3.



The roadmap of Push Chain is structured into three different phases, each of which tackles the problems faced by the apps in that specific segment or niche.

1. **Phase-1:** Consumer-centric apps — enables building true scale apps, supports transaction from any chain, vm for smarter / complex applications, consumer transaction (ordering is not important), fees abstraction - enabling fees from a chain of user choice or from web3 app, wallet abstraction - social login and login from any other chain.
2. **Phase-2:** Enabling seamless interoperability — intermediary phase to enable the universal smart contracts and shared state.

3. **Phase-3:** Universal smart contract and shared state — enabling universal smart contracts making other chain's state accessible on Push Chain and unlocking the era of universal apps.

Phase 1 introduces **consumer transactions** (where order is not important) along with **traditional transactions** (order is important) to enable new innovations to be built along with improving existing innovations of web3**.** It applies new optimization such as **parallel block execution and dynamic sharding** to create the true scaling of web3 apps, enabling read and write capabilities that scale as the number of nodes in the network scales, which provides an infrastructure that can handle the needs of thousands of applications related to speed and scalability.

It further innovates by introducing ability for allowing **any chain to do transaction** on it's chain and leverages **fees abstraction** to ensure that any existing web3 chain user can automatically swap fees from swapper smart contracts present in all supported chains, along with giving the ability to a web3 application or a delegated wallet to cover fees.

It further enables wallet abstraction by allowing any chain user to log in from existing web3 blockchain or using email / social login to provide seamless login and use for a new web3 user or a web2 normie.

It also adds a **Unified User** feature that allows a user to assign multiple EVM as well as non-EVM wallets to **Push ID** ensuring that a user's wallet can be reverse mapped to DID[1]. Because of this, apps can fetch transactions done by the user instead of the wallet doing the same.

All of this is done under a Proof of Stake system that imposes penalization and rewards for nodes to enable a decentralized chain that can be run by anyone in a trustless, permissionless way.

Some of the use cases that can built due to Phase 1 features are:

- **PMF Apps Expanding Their User Base.** Applications that have achieved product-market fit within a specific chain ecosystem but seek to reach users across multiple blockchains are a perfect fit for Push Chain. Examples include prediction market apps that are

[1] https://ethereum.org/en/decentralized-identity/#what-are-decentralized-identifiers

accessible to all users, naming services that resolve to wallets across different chains, and other PMF-driven dapps.

- **Decentralized Social Apps Requiring Fast Finality and Scalability.** Social platforms benefit greatly from Push Chain's fast, sub-second finality and its ability to unify users across blockchains. Push Chain is an ideal platform to build next-generation social apps such as social investing platforms (e.g., GoFundMe or grants), decentralized alternatives to Reddit, Telegram, or TikTok, and other social networks where features like replies, likes, and user connections rely on unordered transactions (consumer transactions).

- **Utility Apps with Universal Reach.** These applications thrive on combining user bases from all blockchains. Examples include email services, notifications and chat platforms, blogging tools, decentralized wikis, and internet archives. These apps benefit from a unified web3 user base, making Push Chain the perfect environment for their growth.

- **Gaming and Degen Apps.** Gaming and degen apps can leverage Push Chain to enable cross-chain participation. Examples include poker platforms, slot games, sports betting, fantasy sports, shared lottery systems like PoolTogether, trading card platforms, and gacha games. Apps in this category benefit from storing intermediate user states or combining user bases across blockchains.

- **Unified Data Layer for AI Agents.** AI agents (autonomous programs acting on behalf of users or protocols) often need to access, analyze, and react to a large variety of onchain and offchain data. Push Chain's tx payload size and dynamic sharding capabilities allow more effective AI agents to be trained and optimized for onchain decision making.

- **AI based User Modeling to improve UX / Context-Awareness**. Using Push Chain's unified user identity (Push ID), AI agents can retrieve a user's aggregated state from multiple chains—assets, past activities, communication history, and more. This comprehensive, chain-agnostic view helps AI agents deliver more contextualized and accurate responses or services.

- **User Settings and Meta Profiles.** While not immediately mainstream, decentralizing the storage of user settings, profiles, and preferences has significant potential. These features represent a step toward a fully decentralized web3 experience, where user data is securely stored and managed.

Phase 2 shall aim to establish seamless interoperability between chains to enable cross-chain features in the apps. The interoperability shall be enabled using an solver network capable of instant cross-chain transactions to ensure better UX. The network shall

be designed to move not just value but also arbitrary payloads that seamlessly transfer smart contract calls to other chains.

This is an intermediate requirement to enable a chain that provides seamless interoperability and hyper-scalability from Phase 1. This further opens up the possibility of building scalable universal apps that aren't tied to one specific chain, are scalable, and provide better UX without forcing app users to deal with any on-chain intricacies.

Some of the use cases that can built due to Phase 2 features are:

- **Unified DEXes.** With bridging capabilities, tokens and stablecoins from any chain can be utilized to create a seamless decentralized exchange (DEX). This onchain, consumer-friendly platform would enable smooth exchanges and transfers across chains.

- **Web3 Portfolios.** By mapping wallets through Push ID and leveraging seamless interoperability, web3 portfolios can track balances and assets across multiple chains, providing users with a unified view of their holdings.

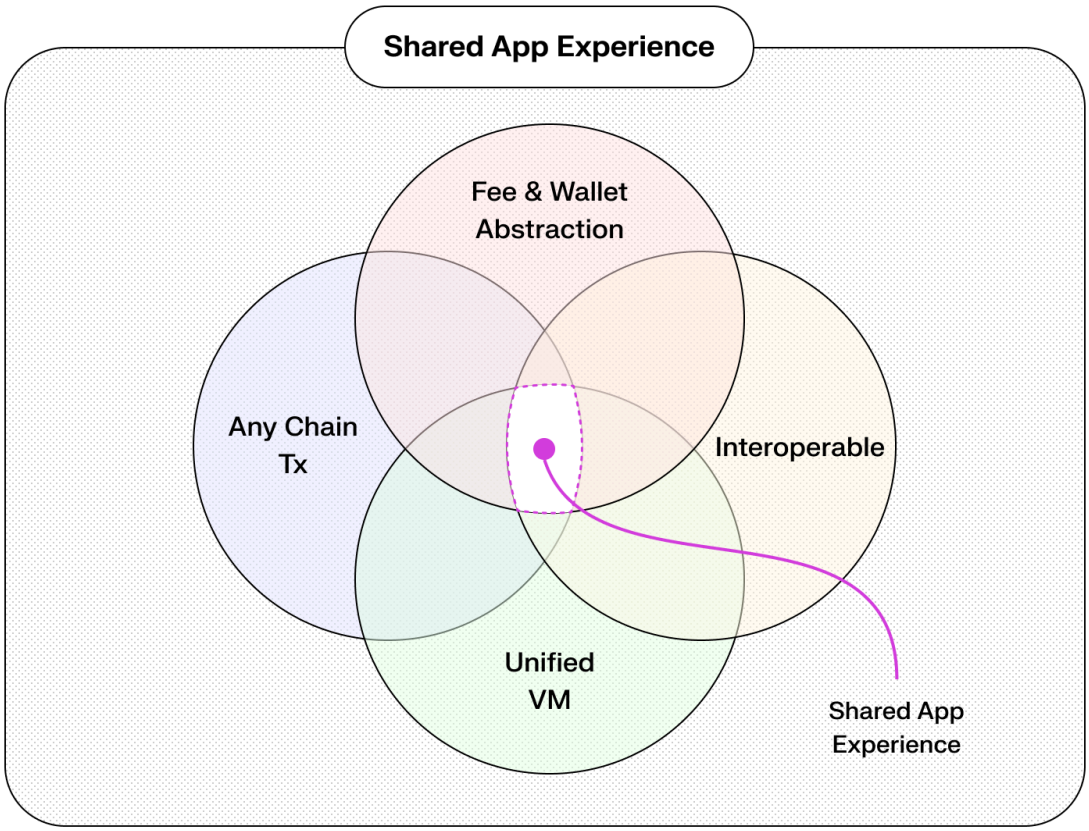
- **Unified DeFi Apps.** Users from any chain can lend and borrow, addressing fragmented liquidity and enabling applications that act as universal liquidity hubs—taking liquidity from one chain and providing it to another.

- **Trading NFTs Across Chains.** NFTs from different chains can be bridged and traded within a single application, further unifying the web3 ecosystem and enhancing accessibility for NFT enthusiasts.

- **Seamless Interoperability for Chain Agnostic AI Actions.** AI agents often need to act across multiple ecosystems—one chain might have liquidity pools, another might have NFT marketplaces, and another might have lending protocols. Using interop and knowledge of multiple user wallets across chains, an AI agent could rebalance a user's DeFi positions on Ethereum, list NFTs on Polygon, and communicate status updates to the user via a messaging app on another chain, all through one unified settlement layer.

More about Push Chain Phase 2 can be found below.

Phase 3 is aimed at bringing the era of **universal apps** to web3. It builds on top of the existing concepts introduced in Phase 1 and Phase 2 and introduces shared state and universal smart contracts to abstract all of the concepts of chains and wallet management away from the user.



Shared state means that Push Chain not only has read and write access to its own state but also has read only access to all the other chain's state.

Combining this with Push ID enables universal smart contracts wherein the smart contracts on Push Chain have access to the state of all of the wallets of the user even if those wallets belong to other chain(s). This enables developers to understand the user and build universal smart contracts to fulfill the user's need irrespective of what chain(s) the user state is on which ushers countless innovations.

The feature opens up the entire web3 userbase to be available for a web3 application enabling shared app experiences no matter from what blockchain the user accesses the

application. It also makes Push Chain the shared settlement layer for all other chains (EVM / Non-EVM).

This enables developers to completely abstract away any learning from the user, creating an era where a user does not use Uniswap “on Ethereum” or Uniswap “on Polygon” or Uniswap “on Arbitrum” — but rather simply **uses Uniswap** for its features**.**

It also ensures that a developer never has to worry about migration away from any L1s, L2s, or L3s as the user is not using their app from a chain-specific view. It is not necessary for the user to know what chain(s) a protocol is deployed on as Push Chain takes care of handling all the other arbitrary logic leaving developers to focus simply on getting users and users to focus only on the features of the application.

Some of the use cases that can built due to Phase 3 features are:

- **Universal Web3 DEXes.** Enable abstracted swaps on any chain by utilizing state information from multiple wallets, facilitating seamless fund movement and swaps.
- **Universal Web3 DeFi.** Provide unified liquidity across EVM and non-EVM chains, delivering seamless DeFi services to users.
- **Universal Web3 DAOs.** Allow DAO voting by aggregating votes across chains for a single user.
- **Unified Web3 Gaming.** Bring players from different chains together for unified tournaments or gaming sessions.
- **Unified Web3 Marketplace.** Abstract NFT trades, enabling seamless swaps across any chain.
- **Universal Web3 Socials.** Enhance web3 social platforms with token gating, NFT gating, or user behavior-based gating.
- **Universal Web3 Analytics and Recommendation.** Enable analytics to track user activity and capabilities, unlocking recommendation engines, AI based decision engines, targeted promotions, and other innovative features
- **AI and Universal Smart Contracts.** AI agents will be able to analyze and leverage states from multiple chains in a unified manner. They will incorporate complex logic that

dynamically adapts to data sourced from different ecosystems. This unlocks even more sophisticated AI-driven services (e.g. universal AI financial advisors, multi-chain governance delegates, cross-chain content recommendation engines) functioning seamlessly in one integrated environment.

More about Push Chain Phase 3 can be found below.

3.1 Enabling true scale, any chain transaction for consumer apps

Consumer apps have the ability to disrupt the entire web3 and how we interact with it. For an infrastructure to support that app ecosystem, it must enable that apps have the following features:

1) **Can be interacted with by any blockchain wallet address**. Users should be able to interact with an app no matter what chain they are on and how they choose to interact.

2) **Can achieve true scale.** There should be linear scaling of read and write throughput as the nodes in the network grow.

3) **Can efficiently retrieve information pertinent to what the app user is consuming.** A user should be able to retrieve transactions based on category wise indexing, enabling apps to quickly fetch and construct interfaces by only accessing transactions that are useful for them.

4) **Can abstract fees.** A non-push-chain wallet should be seamlessly able to pay the fees, either by swapping native tokens of the blockchain it is interacting from, or via delegation or by web3 application apart from the usual wallet pays fees model.

5) Is able to provide **seamless wallet abstraction** - A user should be able to login once and then consume the app without the need to sign again (and again).

   Also, existing web3 users should not be forced to be onboarded to the chain and email / social abstraction should be provided for new users or web2 users (ie: not

forcing user behavior change).

6) Users should be treated as a **Unified User** and not a single wallet address - A user can and most likely have multiple wallets spread across both EVM and non-EVM blockchains. Transactions done through any of a user's wallets should map to a single unified user.

7) All of the features work **without diminishing the security of the chain** - A user should be able to have access to the above features and UX without sacrificing the decentralization ethos or security provided by web3.

3.2 Design Principles

The design of the Push Network incorporates dynamic security assumptions, allowing the protocol to adapt and maintain probabilistic safety properties even when the number of malicious actors exceeds typical thresholds.

This flexibility is crucial for supporting a wide range of applications, from push notifications to decentralized apps (Web3 apps), ensuring both performance and security are not compromised.

Additionally, Push Chain's structure supports dynamic parameter adjustments through participant voting. These parameters include the number of required attestations for block finalization and the average number of validators per block, ensuring the network achieves the highest throughput with the shortest finality.

Push is inspired by the design principles of true scale, ease of use and delivering the best experiences possible to users and developers.

1. **True Scale:** The protocol needs to scale based on the demand of the application and not in a way that is tied to the TPS of different blockchains. It should be able to handle real demand from consumer apps in both read and write throughput without putting a condition on app interaction or how an app wants to use it.

2. **Ease of use:** The protocol needs to be seamless in usability both on the developers side as well as the consumers side. It should learn from the user experiences of

web2 apps and apply them wherever possible but not compromise on security.

3. **Understandability:** The design of the protocol should be optimized to be as simple to understand as possible. Simplifying concepts leads to collaborative efforts and shared thinking, things required for innovation to foster and turn into a perfect product.

4. **Decentralization:** The protocol needs to be decentralized and should be made in a way where compromise or collusion is unsustainable. To create a global machine infra that handles all consumer apps, it needs to be built with a decentralization ethos, ensuring long term survivability.

5. **Sustainable:** The protocol needs to be sustainable enabling users and developers to come and freely use it without worrying about the sustenance of the project.

4. Technical Overview

Note: Some of the technologies are still in the research phase and are open to change based on edge cases that might get uncovered.

This is especially true for Phases that are further away in the roadmap of Push Chain. The whitepaper is always open for RFC (Request for Comments) from the community. Features that are still in active research phase are marked with (*).

4.1 Terminologies

1. **True Scale:** Push defines true scale as read and write throughput linear scalability as the number of the nodes in the network grows. To do this, Push utilizes a number of optimizations.

2. **Dynamic Sharding:** Ability to have sharded chunks that map and maintain the respective state of a range of wallets instead of all wallets, with remapping occurring as the number of nodes are added or removed from the network.

3. **Consumer Transactions:** A consumer transaction can be defined as the type of transaction that doesn't benefit from the strict ordering as seen in traditional transactions. . These can also be categorized as transfers of consumer data like notifications, emails, blogs, etc that do not require a strict order to be executed.

   Since there is no state change for such transactions, they don’t need to change the merkle root of the network and thus can be called **orderless transactions.**

   Examples of these transactions can be email sent from one user to another, posting on social media, writing blogs, creating content, defining user settings for an app, defining user meta-data, storing soft state data for gaming among others.

4. **Traditional Transactions:** Any transaction that requires maintaining a strict order is defined as a traditional transaction. These are primarily financial in nature but can

have arbitrary data with strict ordering as well.

5. **Fees Quota:** Fees Quota refers to an allowance package that users purchase using native tokens on the Push Chain network. This quota is used to cover the transaction fees associated with various on-chain actions.

6. **Swapper Smart Contract:** These are swapper / fees contract deployed on all supported chains to enable fee abstractions. They allow a user of any chain to send fees to Push Chain by just swapping the native token of that chain from our swapper contract enabling seamless flow.

7. **Push ID (Unified User):** Push ID represents a unique user identifier(hash of master key) that can be mapped to multiple EVM and non-EVM wallet addresses. This enables wallets to be reversely mapped to Push ID to get transactions for a unified user instead of a single wallet attached to them.

8. **Slashing:** Slashing is a security mechanism used by the Push Chain to penalize validators or participants who act maliciously or fail to fulfill their responsibilities, such as validating transactions correctly. When slashing occurs, a portion of the participant's staked tokens is forfeited and removed as a penalty, discouraging harmful behavior and ensuring the integrity of the network.

4.2 Key Management

To enable the Push unified network where users can interact with any chain securely, benefit from wallet abstraction, and be identified as a **Unified User**, Push proposes a new Key Management architecture.

4.2.1 Objectives

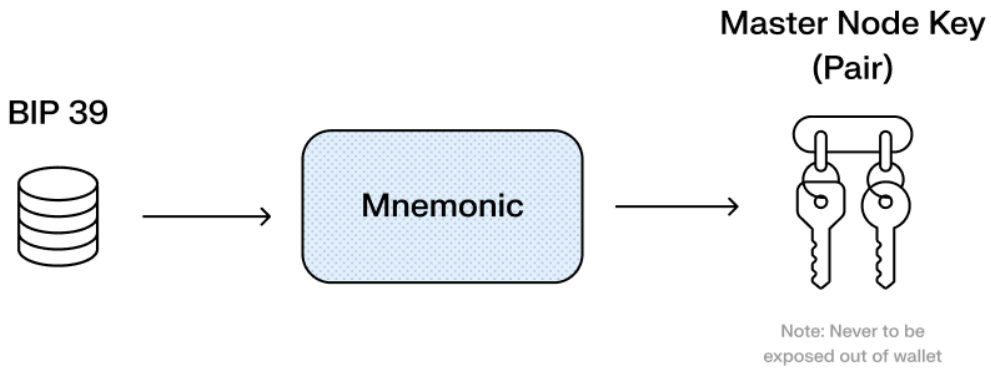
- **Security**: Ensure the highest level of security for users' keys, safeguarding them from any unauthorized access or breaches.
- **Usability**: Provide a seamless and intuitive user experience, making key management straightforward and user-friendly.
- **Interoperability**: Facilitate smooth interactions across multiple blockchain networks, ensuring broad compatibility and integration.

- **Wallet Abstraction:** Simplify blockchain interactions by abstracting wallet operations. Support social logins, phone or email-based authentication, and secure recovery mechanisms, enabling secure and intuitive access control for users.
- **Network Abstraction:** Enable seamless interactions with multiple networks by dynamically generating keys for different blockchains from a single unified wallet, ensuring compatibility across ecosystems.
- **Scalability**: Maintain high performance while supporting an expanding user base and an increasing number of keys.
- **Encryption**: Offer native support for encrypting data for recipients, ensuring secure and private communication.

4.2.2 Key Generation

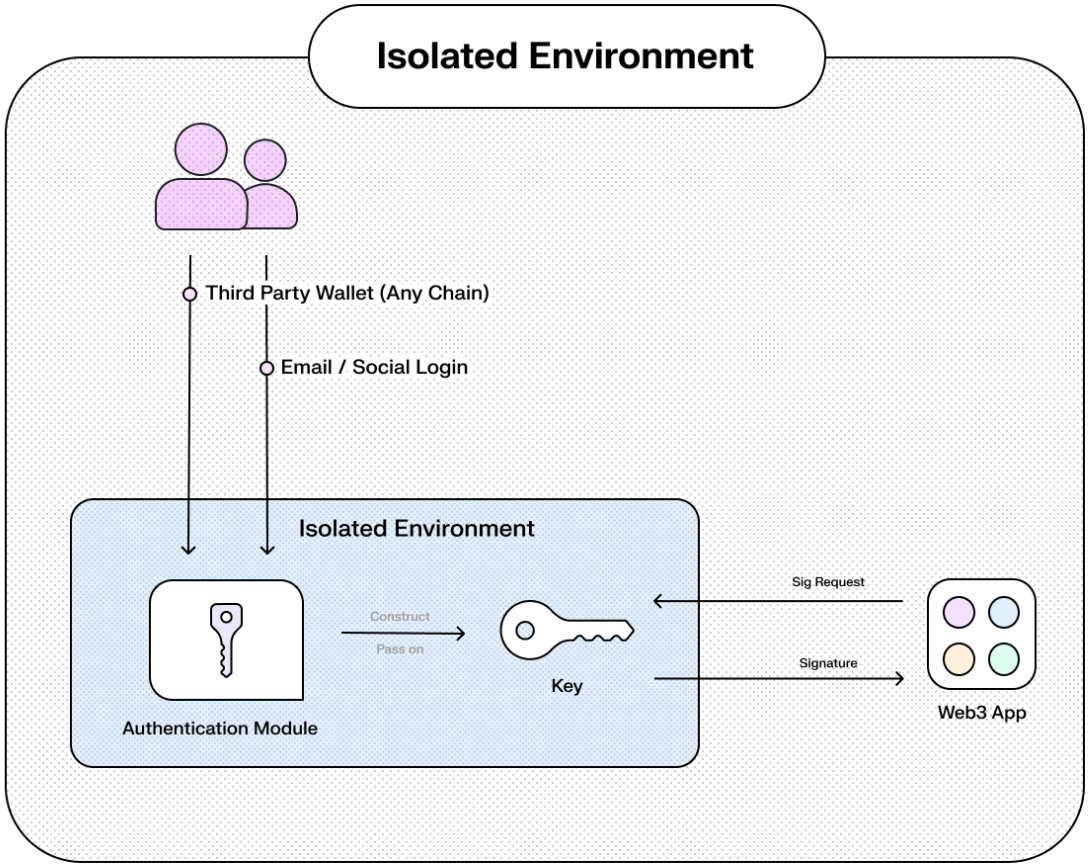
Keys are generated using mnemonic phrases based on BIP-39 and hierarchical deterministic (HD) wallets based on BIP-32.

This ensures that keys can be easily backed up and restored using a simple mnemonic phrase, which enhances security and user experience. Additionally, HD wallets also supports derivation of infinite keys used for multiple purposes in the Push Network.



4.2.3 Key Interaction

Keys are managed within an isolated environment to prevent direct exposure. This secure environment can take various forms, such as an iFrame, browser extension, or mobile application. Keys will never leave this environment similar to any other web3 wallet.



The utilization of an isolated environment significantly enhances security by preventing unauthorized access to the keys, thereby protecting users' assets and sensitive information.

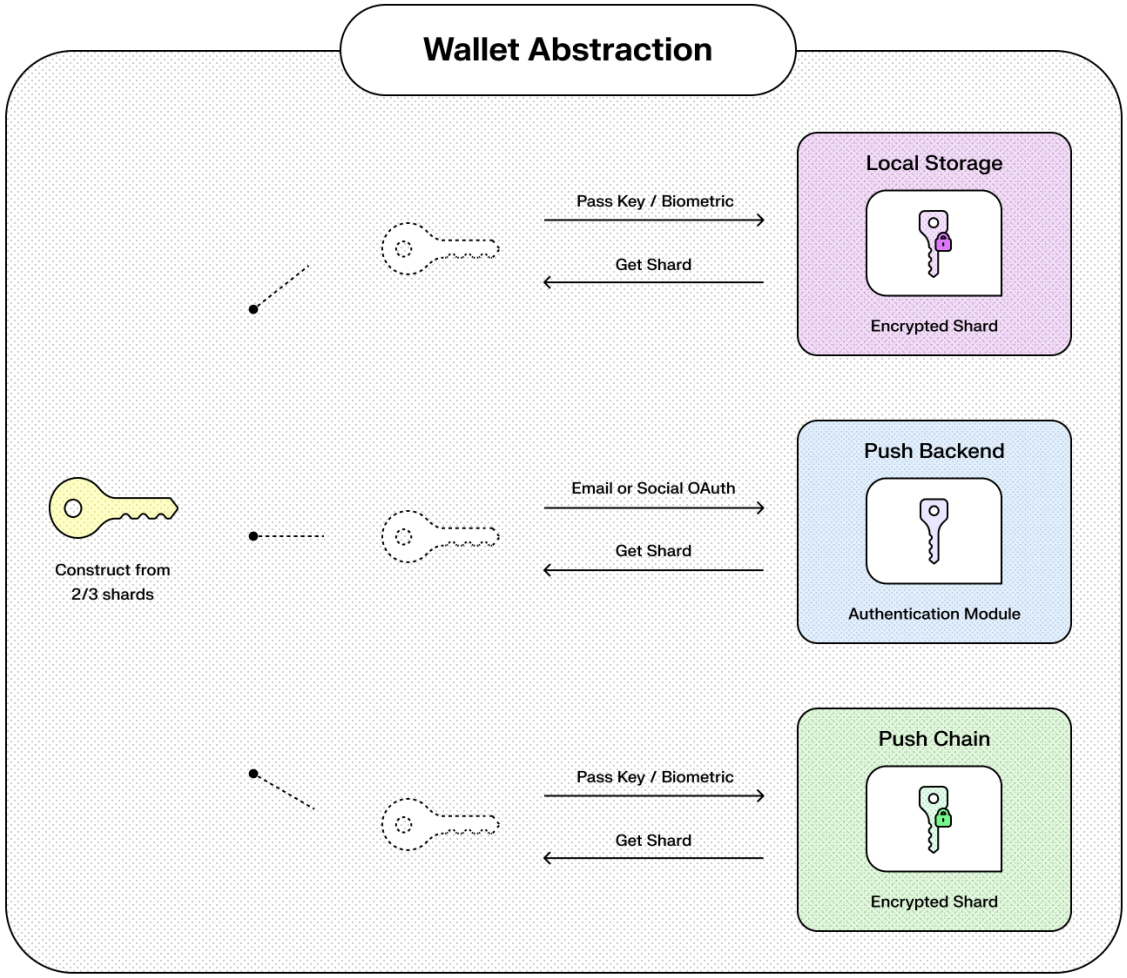
4.2.4 Enabling Multi-wallet Account Management

Push Chain’s key management empowers users to link or unlink multiple blockchain accounts to their unified profile, enabling a seamless and comprehensive view of their assets and activities across various networks. This approach simplifies account management by centralizing interactions within a single interface, eliminating the need to juggle multiple wallets independently.

In addition to providing a unified view, linked accounts can also support advanced features such as gas fees payment from any linked account or distribute costs across accounts, offering flexibility and cost efficiency.

4.2.5 Simplifying UX with wallet abstraction

Managing mnemonics for wallet custody can be challenging and confusing for users, especially those new to blockchain technology. The process of securely storing and retrieving a mnemonic phrase introduces significant complexity and can lead to user errors, such as loss of access or accidental exposure of sensitive information.



Wallet abstraction addresses these challenges by removing the need for direct mnemonic management. Instead, Push Chain allows users to authenticate using familiar methods like social logins, email, or phone numbers. This system abstracts away the complexity of mnemonic custody by securely managing a master key pair on behalf of the user, while still ensuring they retain ultimate control.

The Push Chain wallet can dynamically derive blockchain-specific keys from the master key pair, enabling seamless interaction across multiple blockchain networks. By simplifying

wallet operations and providing a unified experience, wallet abstraction reduces friction, enhances usability, and makes blockchain technology more accessible without compromising security.

4.2.6 Data Encryption

Transaction data can optionally be encrypted for participants, such as the sender and recipients, to ensure secure transmission of private information. The Push wallet utilizes the master key pair to derive public keys for various blockchains, which can be shared to facilitate encryption for onboarded users.

For non-onboarded users with external blockchain accounts, encryption poses a unique challenge. Since these users will not have their encryption keys managed within the Push unified account. This can include:

- **Network Custodial Encryption:** Temporarily encrypting data on the Push network using a network-managed key, ensuring secure delivery to non-onboarded users.
- **Trustless Encryption Mechanisms:** Exploring trustless encryption solutions that allow secure communication without relying on custody, potentially leveraging zero-knowledge proofs or other decentralized methods.

These approaches aim to provide flexibility and inclusivity, ensuring that even non-onboarded users can securely participate in encrypted communication.

4.2.7 Key Storage

- **Mnemonic Custody**: The mnemonic is divided into three shards, enabling a 2-of-3 recovery mechanism:
    - User Shard: Held securely within the user's secure environment.
    - Push Shard: Encrypted and stored on Push servers, accessible only through user authentication.
    - Chain Shard: Encrypted with a user-owned passkey and securely stored on the Push chain.

  This sharding mechanism can be discarded for advanced users to support full user custody, ensuring flexibility for varying security needs.
- **Web3 Account Linking:** Pre-existing accounts on other blockchains can be linked to the Push wallet using a link account category transaction, providing users with a unified view of their blockchain accounts.

- **Unified User Details Storage**: The master public key and linking details for associated accounts are stored on Push storage nodes via an init transaction, designed to be open, well-indexed, and optimized for fast querying.

4.3 Transactions

A transaction can be defined as a digitally signed instruction that securely moves data or value from source to destination. Each transaction is verified by network participants, ensuring its authenticity and preventing double-spending.

The Push Network is designed to facilitate a wide range of consumer transactions, providing a robust and scalable platform for decentralized applications. The transaction model in the Push Network is tailored to support high-throughput, low-latency operations, ensuring that various applications such as email, social media, blogging, and notifications can interact seamlessly.

4.3.1 Transaction Payload*

A typical transaction payload for consumer transaction in Push looks like the following:

```javascript
{
       "category": 'email',
       "transaction_type" : 0,
       "salt": '0x123'
       "source": 'ETH_SEPOLIA'
       "signature": '0xabc',
       "recipient":[eip155:1:0xD8634C39BBFd4033c0d3289C4515275102423681,
       eip155:97:0xD8634C39BBFd4033c0d3289C4515275102423681...]
       "fees": 1000
       "data" : '0xabc....data'
 }
```

Definitions of parameters in a transaction payload:

1. **category** represents the type of the consumer transaction. Since there could be a wide range of different consumer transaction, this field helps in identifying specific types of transactions in the network. The category field can include values like *email, blog, notifications etc.*

2. **transaction_type** indicates the type of transaction being a **traditional** or a **consumer transaction**. Considering the fact that Push Chain will support both financial (traditional) and consumer transactions (non-financial / mixed financial), the field helps in differentiating between the two so that the node operators can process accordingly.

3. **salt** is an important field that helps ensure that two transactions never lead to a similar transaction hash, thus helping in creating a unique identifier for each transaction.
   Note: *A salt must not be confused with a transaction nonce. Nonce imposes a strict order in transaction execution while salt only helps in creating a unique identifier. The need for strict order is not imperative for consumer transactions.*

4. **signature:** This field includes the digitally signed signature of the sender of the transaction.
5. **recipient:** This field includes the list of recipients for a transaction. The unique aspect of a Push transaction is the fact that there can be more than one recipient at a time belonging to one or more chains. Additionally, it must be noted that all recipients are to be included in CAIP-10[2] format.

6. **fees** is a field that indicates the amount of fees required to complete a transaction.

7. **data** is a crucial field as it contains the encoded form of all the important messages (payload data) to be transferred from source to destination.

4.3.2 Transaction Lifecycle

The life cycle of these transactions involves several stages, ensuring security, efficiency, and seamless operation. Here is a detailed overview of each stage:

[2]CAIP 10 Blockchain specific address:
https://github.com/ChainAgnostic/CAIPs/blob/main/CAIPs/caip-10.md

1. Initiation

The life cycle of a consumer transaction in the Push Chain Network begins with the initiation phase. The user creates a transaction by specifying the required fields.

2. Signing

Once the transaction is initiated, it must be signed to ensure authenticity and integrity. The transaction payload is encoded using **Protobuf**, a serialization method that ensures a compact and efficient data format.

The user signs the encoded transaction payload with their key. This digital signature is then attached to the transaction payload, providing a mechanism for verifying the sender's identity and ensuring the transaction has not been tampered with.

3. Broadcasting

The signed transaction payload is broadcast to the Push Chain Network. This involves sending the transaction data to all nodes in the network, making it visible to validators and other network participants.

4. Validating (Pending State)

The broadcast transaction enters the validation phase, where it is checked for correctness and authenticity. This involves verifying the digital signature, checking the transaction format, and ensuring it adheres to network rules and policies.

During this phase, the transaction is in a pending state. Users can view their transaction as pending, indicating it is being processed and validated by the network nodes.

5. Confirmed or Reverted

5.1 **Transaction Confirmation:**

- If the transaction passes all validation checks, it is confirmed and added to the network.

5.2 **Transaction Reversion:**

- If the transaction fails validation (e.g., due to an invalid signature, insufficient fees, or incorrect data format), it is reverted.

6. Successful

The final phase marks the successful completion of the transaction.

- Once confirmed, the transaction reaches the successful state. This indicates that the transaction has been fully processed by the network and the expected outcome is now achieved.

4.4 Transaction Fees: Fees Abstraction and Quota

A key feature of Push Chain is Fee Abstraction. This eliminates the friction of requiring to hold native gas strictly in the wallet in use as holding and managing native tokens adversely affects the UX of using a web3 app. adversely affects the UX of using a web3 app.

Furthermore, a sustainable decentralized network needs to be present in order to ensure long term viability of the project. To do this, we propose including a mechanism for having Fees quota.

A user is able to purchase fees in bulk that can then be used by them, delegated to other users / wallets or can even be paid by web3 applications. The key idea behind locking fees in bulk is to ensure that the transaction coming from whatever blockchain network is not limited by the speed of that particular blockchain's TPS.

Additionally, there is a fallback mechanism that does a graceful conversion using swapper contracts which takes the native token of a specific blockchain and swaps it for Push Chain's fees.

This ensures that in the event a different chain user doesn't have enough fees, even then the network is still able to fallback and use that particular blockchain's native token for the conversion and eventual processing of the transaction.

This mechanism ensures that running out of quota is not a barrier to transaction creation, providing flexibility in payment methods.

4.4.1 Calculation of Fees

In a consumer app, the fees of a particular transaction are dependent on the amount of data that is sent with it. It is therefore important for a partial and unbiased network to have dynamic fees based on the size of the data field.

While the specifics around fee calculation are currently being researched, it's already understandable that the size of the entire transaction will play an important role in determining a base fee for a specific transaction.

4.4.2 Fee Scalability*

One of the key features that Push Chain brings is true scale with dynamic sharding, however this also means that fees can’t be charged in conventional ways as multiple validators are assigned to process transactions.

To ensure that scalable design persists even with addition of fees, Push Chain delegates the role of handing fees of wallets to their specific shard of storage node. A transaction that lands to a validator goes through pre-checks, one of which is checking the shared storage node where that wallet resides to check if the wallet fee balance is still proper and then proceeding with the transaction.

This design decision enables that fees are also scaled with the sharded architecture of Push Chain. Since the validation of transactions are done in parallel, having shard of the storage node (along with quorum) processing the fees and passing the pre-transactions checks removes the bottleneck of fees getting maintained by validators as a global state which ensures that the read and write scalability of the chain is able to still work in the intended fashion.

We also propose validator and specific storage nodes to share a part of the fees in the model as this will ensure that the validator and storage node shards that are picked for processing transactions that land to them are at their most honest behavior as they earn a part of the fees.

4.4.3 Built-In Fee Delegation Mechanism

Consumer apps might want to design systems that make a user pay for transactions or have an fee-abstracted(freemium) approach to certain actions that are done by the user.

For instance, in order to provide smooth onboarding for all of its users, an application might not want to have the added hindrance of users paying any transaction fees.

To accommodate this behavior, the Push Chain has an additional feature that allows transactions to be submitted with a delegate wallet signature enabling apps to build consumer behavior and charges based on their own requirements.

To enable this behavior, transactions can also carry delegated wallet signatures along with other fields in their payloads, which will allow validators to validate and utilize the required fields to eventually claim fees.

Furthermore, smart contracts can also be used to maintain Fees quota and to delegate those quotas to other wallets. Validators should be able to listen in and claim those quotas as and when they want to by submitting these signatures to the designated smart contract involved.

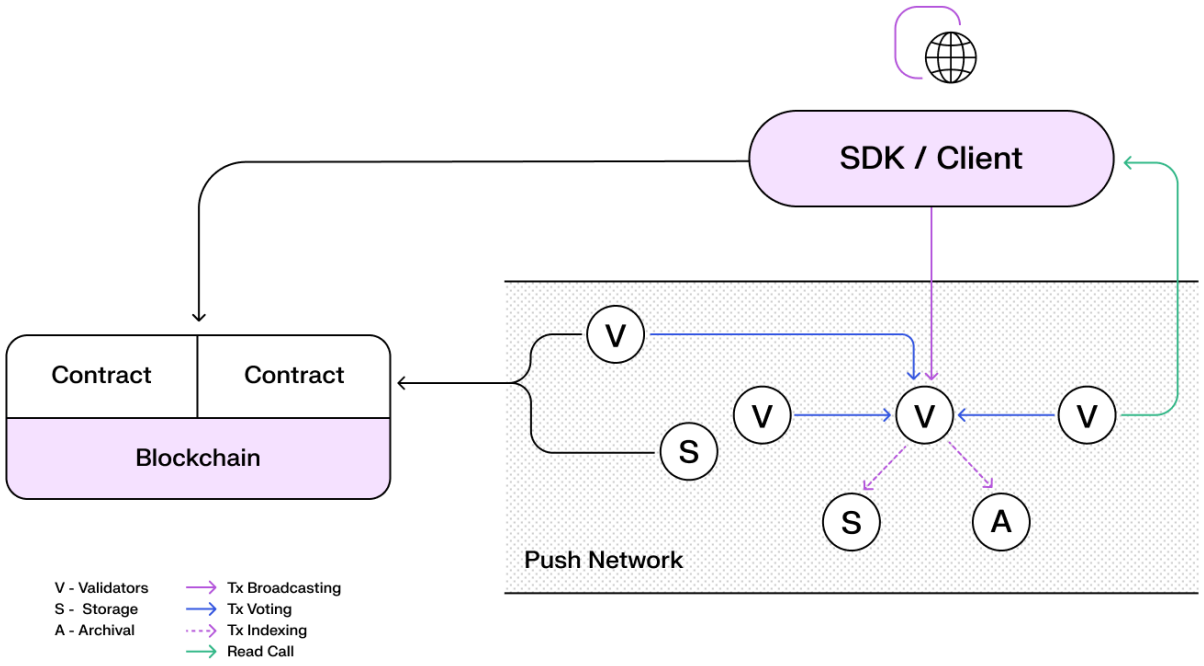
4.5 Network Design

Push is a Proof of Stake (PoS)[3] chain. PoS is a way to validate correct work and transactions in a network with penalization and rewards built in to ensure that anyone can run the network in a permissionless way by staking some amount of tokens.

PoS rewards honest validators for their work and has a mechanism for bad actors or malicious nodes to be punished for submitting false data. The software that is run by anyone to enable this peer to peer mechanism is usually referred to as nodes or sometimes as validator nodes.

Push Chain leverages different types of nodes to enable PoS. Each of the nodes have different roles and responsibilities while a governing smart contract enables peer discovery and handles dynamic sharding, rewards, slashing, etc.

[3] Proof of Stake Consensus: https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/



4.6 Nodes Overview

Push chain relies on different nodes that are tasked with specific operations to enable peer discovery, dynamic sharding, storage, etc.

4.6.1 Validator Nodes

These Nodes are responsible for validating transactions that are generated on the Push Network. Validator nodes ensure the validity of a transaction that is generated by validating the signer of the transaction. They also ensure that either the wallet that generates the transaction or a delegate has enough fees to cover the transaction.

These nodes are the ones that form the gateway between the network and any user that wants to access them. Other nodes usually will interact with validator nodes but generally speaking, validator nodes don't interact with the outside world.

4.6.2 Storage Nodes

These nodes are responsible for storing a transaction along with its payload data, They are different from validator nodes as their primary responsibility is to store data in a sharded fashion.

The transaction data after it has been validated and attested by validator nodes lands over here.

4.6.3 Archival Nodes

These nodes store all the transactions that are processed on the Push Network. Their role is to enable users to access the entire logs of transactions that are processed if needed and also to provide web socket functionality for the streaming of transaction data as needed.

They are useful in the cases of blockchain explorers or in the cases where an entire network’s activity needs to be fetched.

4.7 Nodes Discovery and Selection

4.7.1 Node Discovery

Node discovery is an important aspect of the Push Chain, ensuring that nodes can efficiently locate and communicate with each other to maintain the network's integrity and performance.

Every node in the Push Chain network must first register with the network’s smart contract during its setup. This registration process allows the network to maintain an up-to-date record of all active nodes. During registration, each node provides a public HTTPS JSON-RPC API endpoint. This API endpoint is essential for enabling external interactions with the node, such as transaction submissions and queries.

Once registered, nodes can be located using SDK code by performing a read operation on the smart contract. This operation retrieves the necessary details, such as the node’s public API endpoint, which can then be used to interact with the node. This process applies to both validator nodes and archival nodes.

It must, however, be noted that storage nodes, which are responsible for storing data rather than processing transactions, do not serve user API requests and are not involved in direct user interactions.

While all nodes are registered and can be discovered via the smart contract, validator nodes have a unique requirement for discovering and communicating with each other.

Validator nodes must regularly interact to perform critical network functions, such as signing blocks, executing slashing protocols to penalize malicious behavior or the creation of randomized tokens. This interaction is facilitated through a secure and efficient node discovery process.

4.7.2 Randomized Transaction Submission

A crucial security mechanism of the Push Chain is the randomized selection of validators for transaction validation or block production. This mechanism leverages cryptographic randomness to ensure an unpredictable selection of validators & attestors to enhance the network's resilience and fairness.

To send a transaction to the network, the SDK/sender must attach a randomly generated token number to the transaction. To fetch this token, the SDK can request the same from any validator. Once received, the SDK attaches this token to the transaction and sends it to the network. The network then uses this random token to cryptographically decide the specific validator to verify the particular transaction for further processing. Additionally, the validator also randomly decides the validator set responsible for attesting to the transaction.

Mentioned below are the detailed explanations of the multi-level randomization involved in the transaction submission procedure.

The Push SDK selects a random validator (VR1) to generate a special signed (RANDX) hex token. The generation is performed as follows:

- **Generation of Random Token:**

  The network is designed to ensure that a random token cannot merely be generated by a single validator. Instead, it must be a combined effort as every random token is ideally a collection of digitally signed pieces of random numbers provided by specific validators in the network.

  To determine the exact number of such pieces required to form the token, the Validator.sol smart contract defines a parameter called *required_random_number.* This indicates the number of required random peers whose signed values each validator must hold, at a given time. Additionally, the contract also defines an

expiration_time parameter that defines the expiry time for these tokens which helps ensure that only freshly created tokens can be used for the creation of random tokens.

As previously mentioned, validators in the network periodically ping their peers to collect and exchange signed random values. The collected value from each validator is a combination of the random value itself and the validator's signature, ensuring it can be verifiable.

```
Unset
      signedRandomValue = value + validatorSignature
```

As a result, at any given moment the validators should have a collection of signed random values by their peers, i.e., **signedRandomValue[].**

Once a validator has accumulated the necessary signed random values, it can generate the random token. This token is derived from the network's collective randomness using the following formula:

```
Unset
      RANDX_Token = XOR(sha(signedRandomValue[i])) for all i
```

- **SDK Requests for Random Token**

  As previously mentioned, before sending any transaction to the network, the SDK requests for a random token that must be attached with the transaction being sent.

  Now, since a random token value has already been generated (as mentioned in step 1), any validator can provide this random token to the SDK.

  The SDK then attaches this random token to its transaction payload and sends the transaction to the network.

- **Randomized Selection of Validators**

  This random token attached to the transaction plays a crucial role in randomly deciding the validator that will process the transaction. With this token, the SDK deciphers the validator id it must send this transaction to.

```
Unset
validator_id = f( RANDX_TOKEN, contract_defaults)

//Output
validator_id = v1

where contract default parameters could be:
a. min required quantity of random numbers,
b. expiration time,
c. minimum number of peers that acknowledge the validator's
online status.
```

  Upon receiving the transaction, the validator checks the random token in the transaction payload. It validates the token to confirm that it has been correctly assigned the role of handling this transaction. This validation step ensures that the transaction is processed by the correct validator as dictated by the network's randomized assignment process.

  In case the validator identifies that it has been wrongly chosen, it can reject the transaction.

  Once ensured that the correct validator has been chosen for transaction validation, the validator now creates a similar randomized process for creating a list of validators who must attest to this transaction.

```
Unset
vlist = f(RANDX_Token_2nd, contract_defaults)

// Output
vlist = V2, V50, V11, V8, V38, V17, V29
```

This list of validators is now responsible for attesting to the transaction created and verifying its **legitimacy.**

To summarize, the entire process adds a security layer with the inclusion of randomization at multiple levels in the process, i.e.,

- **1st Level:** The first randomization starts with the SDK choosing any random validator to request for a token to be attached to the transaction.
- **2nd Level:** The generation of the token using digitally signed pieces of random values from multiple random validators instead of relying on any single actor.
- **3rd Level:** The random selection of a specific validator to create the token to be attached in the transaction occurs.
- **4th Level:** The final level of randomization determines which specific validator (VR2) will process a transaction and which validator set will act as attestors to verify the transaction.

4.8 Block Production and Block Attestation

Each validator in the network has the capability to produce blocks, provided there are incoming transactions. When a validator receives a transaction request, it collects all transactions into a local pool. At regular intervals, a block is produced from these transactions and is subsequently published into a virtual queue for storage nodes.

When a validator is tasked with proposing a block (B1), it selects transactions (T1, T2... TN) from the local pool and appends its signature (VS). The structure of a proposed block is as follows:

B1 { T1, T2, ... TN, VS1 }

The attestation process involves a predefined list of validators assigned to verify the block. This attestation list is determined using a series of random numbers.

The Validator.sol smart contract specifies various properties such as the number of required random numbers, their expiration time, and the number of peers that must notice the validator's presence online.

A special function generates a list of validators to attest to the block. This function ensures that given the same set of random numbers (RANDX) and smart contract constants, it yields identical results across all nodes.

For a network with N validators, the function selects M validators (M << N). The selection process is defined as:

vlist = f(RANDX, contract_defaults)

Consequently, after block production, the attestation list is predefined by a collective random number generated by the peers.

An example of an attestation list might be:

vlist = V2, V50, V11, V8, V38, V17, V29

All validators in the attestation list are responsible for verifying the proposed block. They each cast a vote on the validity of the block. The block's validity is then determined based on the majority vote of the attestors.

This majority-based consensus ensures that only valid blocks are added to the protocol, maintaining the integrity and reliability of the network.

4.9 Block Verification and Finality

A block is deemed final immediately after the attestation process is completed. It is then published into the virtual queue of every validator that has signed the block, whether as an attestor or a producer.

Storage nodes will read each validator's queue and index the block only if it can be verified (i.e. if it meets the required number of signatures).

This threshold is dynamically defined by the smart contract involved and adjusts as the network scales.

5. Sharding

Once a block is attested to and admitted into the network, Push employs a sharding mechanism to distribute transactions based on their source and destination addresses. This method enhances parallel processing and storage efficiency across the network.

Write Operations and Indexing

For a write operation to be considered successful, at least 51% of the storage nodes assigned to a specific shard must index the transactions within the block. This process ensures reliable data storage with redundancy and fault tolerance:

1. **Block Admission**: Attested blocks are accepted into the network.
2. **Transaction Sharding**: Transactions are divided into shards.
3. **Indexing**: Storage nodes index the transactions.
4. **Consensus for Write**: Success is achieved when 51% of nodes in the shard index the transactions.

Read Operations and Data Consistency

For a read operation to be successful, 51% of the storage nodes in a specific shard must consistently return the same data. This ensures data accuracy and reliability:

1. **Data Request**: A read request is made for shard transactions.
2. **Data Retrieval**: Nodes return the requested data.
3. **Consistency Check**: The returned data is compared.
4. **Consensus for Read**: Success is achieved when 51% of nodes consistently return the same data.

5.1 Dynamic Sharding

Dynamic sharding is a mechanism designed to dynamically allocate and reallocate data across storage nodes to ensure optimal distribution and replication. This process ensures that as nodes join or leave the network, the data is redistributed efficiently, maintaining a balanced load and minimizing operational costs.

Here is a brief explanation of how the algorithm supports the workflows:

When a node n_i joins a cluster, where every node is responsible for X vshards of data, we start with M vshards (a predefined value = 32).

The algorithm should ensure that:

- The network maintains exactly the specified replication factor for the total number of data chunks across all nodes.
- If a node joins or leaves, the data is redistributed, aiming to minimize gas costs.
- The replication factor can increase (up to the total number of nodes) or decrease to 1.
- Redistribution should affect the minimum number of nodes possible.
- Redistribution should favor acquiring fresh shards instead of old ones.
- The difference between the node with the most vshards and the node with the fewest vshards should be as close to ±1 as possible.
- Registration of the first N nodes will raise the replication factor to rfTarget value which is 7.
- An event is emitted indicating which nodes have been modified.

Example:

Consider the following example with the parameters:

- `maxshards = 6`: A shard is a data range hosted by a node in the database.
- `rf = 1`: Replication factor, indicating how many copies should exist on different nodes.
- `nodes = 1`: Initial node count.

Initial State:

- `shardCount = 10, rf = 1, nodeCount = 1`
- 1 => 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

After Adding a Node:

- `shardCount = 10, rf = 1, nodeCount = 2`
- 1 => 1, 2, 3, 4, 5
- 2 => 6, 7, 8, 9, 10

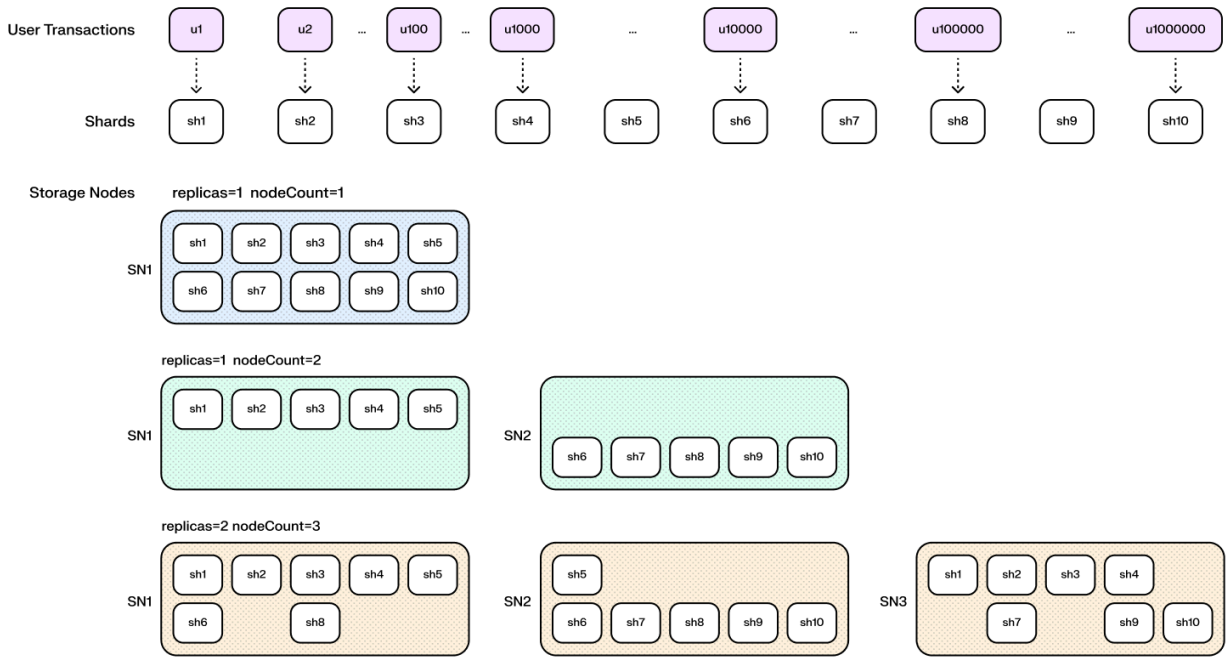
After Increasing the Replication Factor:

- `shardCount = 10, rf = 2, nodeCount = 3`
- 1 => 1, 2, 3, 4, 5, 6, 8
- 2 => 5, 6, 7, 8, 9, 10
- 3 => 1, 2, 3, 4, 7, 9, 10

In this example:

- Initially, with one node, all shards are held by that single node.
- When a second node is added, the shards are evenly split between the two nodes.
- When the replication factor is increased and a third node is added, the shards are redistributed to meet the new replication requirements while maintaining balance across the nodes involved.

These examples illustrate how the algorithm dynamically adjusts to changes in node count and replication factor, ensuring efficient and balanced shard distribution.



Testnet values: vShards: 32, 20+ storage nodes, replication factor = 7,

As can be seen, the network becomes more robust in read and write throughput as the number of nodes increases.

6. Node Management

Node registration, management, dynamic sharding, and data replication are all done through Push's governing smart contract - PushNodes.sol[4]. All nodes listen to PushNodes.sol and update their state accordingly to the state and settings present in the smart contract.

This smart contract is responsible for a number of actions:

- Assisting nodes in peer discovery for all types of nodes
- Maintaining the lifecycle of each node admitted which can be:
    - **OK**,
    - **Reported**,
    - **Slashed,**
    - **BannedAndUnstaked** or
    - **Unstaked**
- Signature validation before moving the lifecycle of the nodes
- Managing and dynamically increasing the **Replication Factor (RF)** based on the **Node Count (NC)**
- Managing and dynamically increasing the **Shard Count (SC)** and the mapping of different shards to data chunks based on the **Node Count (NC)** and the **Replication Factor (RF)**

6.1 Node Lifecycle Management

Any nodes that are keen on participating in the network have to interact with Push's governing smart contract which assigns them one of the following lifecycle values - **OK**, **Reported**, **Slashed, Banned** or **Unstaked.**

These lifecycle values play an important role in rewarding, penalizing and kicking nodes off the network.

Lifecycles are only changed when other nodes complain about a node's dishonest behavior and submit their votes along with the complaint.

[4] Push Node Smart Contracts: https://github.com/push-protocol/push-node-smart-contracts

6.1.1 Admitting nodes - Lifecycle (OK)

Any node that freshly enters in the network is marked as **ok** in their lifecycle which simply indicates that the node is not marked by any bad behavior.

For a node to be admitted to the network, it needs to invoke a function on PushNodes.sol and provide the following information:

- Provide a defined amount of Push Chain's native token for staking
- Provide node information
- Provide nodeApiBaseURL (peer discovery)
- Provide an EVM wallet (for rewards and to sign / attest for reporting, slashing or kicking)

```
JavaScript
struct NodeInfo {
        // the owner of the node;
        // the one who provides PUSH tokens for the node staking
        // also this address is used for unstake
        address ownerWallet;
        // node eth wallet, which is used mostly for private key signatures for node
communication
        address nodeWallet;
        // node type; currently we support only 3 types (V,S,D)
        NodeType nodeType;
        // 'staked' PUSH tokens for this node (18 digits)
        uint256 nodeTokens;
        // rest api url for invocation
        string nodeApiBaseUrl;
        // counters for report, slash, ban
        NodeCounters counters;
        // worst case status, at least 1 report or slash leads to status reported or
slashed
        // lifecycle: OK -> reported -> slashed -> banned|unstaked
        NodeStatus status;
    }
```

6.1.2 Reporting nodes - Lifecycle (Reported)

Before a node is slashed, it is marked as bad as a way to indicate that a node is not performing well. It is the first step to let the node know that they need to correct their behavior or else they will be slashed and eventually kicked from the network.

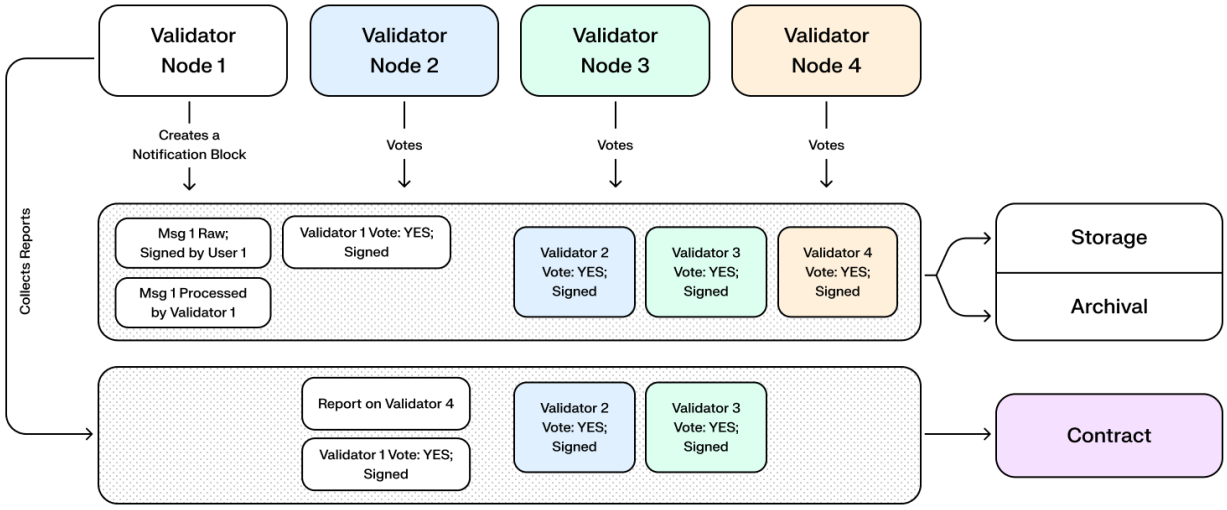
Marking them as bad behavior instead of instant slashing is based on assuming that the node is an honest actor but is not able to perform due to local conditions (bad network, storage, etc).

The definition of bad behavior varies depending on the type of node. For a validator node, bad behavior might involve including incorrect transactions in a block or failing to participate in voting on transactions or reporting false incidents. In contrast, for storage nodes, bad behavior can be characterized by the inability to index and retrieve transactions for the shards they are responsible for.

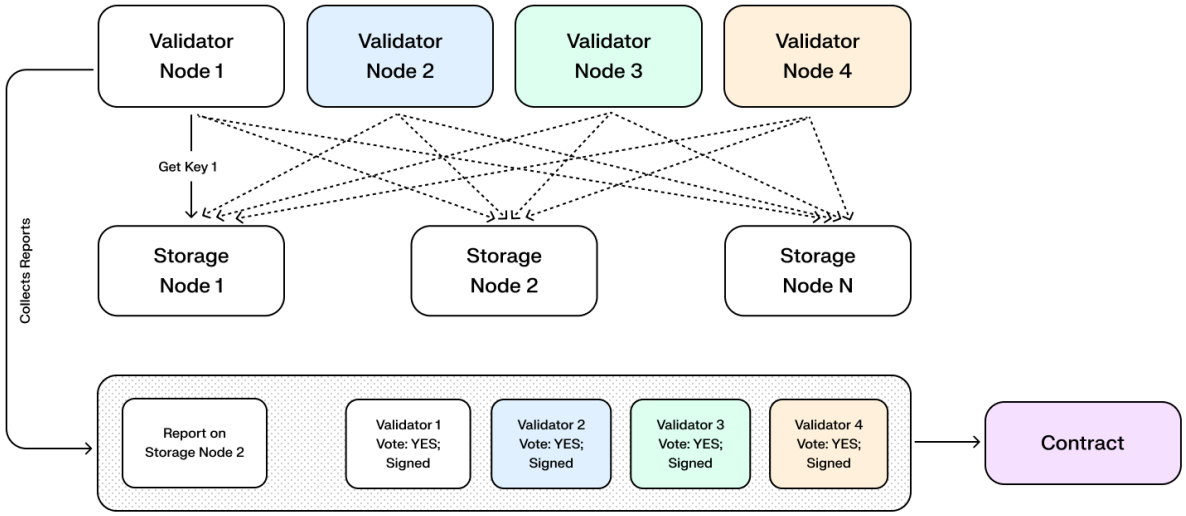
However, if the behavior of the node remains inadequate leading to more complaints, it could lead to the votes count being changed to `REPORTS_BEFORE_SLASH_V_` or `REPORTS_BEFORE_SLASH_S_` in `PushNodes.sol`. In such cases, the node is moved to being slashed.

6.1.3 Slashing nodes - Lifecycle (Slashed)

Slashing a node means taking a percentage of their stake out owing to the repeated bad behavior of the node. Slashing can occur based on invalid data for both validators as well as storage nodes.



Example of validator node slashing



Example of storage node slashing

When a node is slashed, the nodes that complained about the slashed node get awarded the slashed amount.

Each slash of the node results in an incrementing counter of SLASHES_BEFORE_BAN_V_ (_S) in PushNodes.sol.

6.1.4 Banning nodes - Lifecycle (Banned)

Once the threshold for a certain amount of slashes has been reached for a node, then the node is banned from the network and kicked out. A major percentage of the stake owned by the culprit node is slashed as well.

The slashed amount that is taken from the malicious node is also shared between the nodes that voted on the final complaint. This marks the end of the cycle for the node that got kicked out.

6.1.5 Exiting nodes - Lifecycle (Unstaked)

There is another way by which the node can choose to gracefully end their life cycle in the network and that is by choosing to unstake their tokens and withdrawing from participating in the network.

If the node does exit gracefully then the tokens that they have staked are returned back to them as they simply don't want to be a part of the network instead of displaying any malicious or bad behavior.

6.2 Rewards and Penalizations (Slashing)

To incentivize nodes to participate in the network and to align good behavior, we need to have mechanisms that reward / penalize them. This will, in turn, enable a sustainable, long-term network.

6.2.1 Node Incentives

6.2.1.1 Node Rewards

Each type of node benefits from rewards that are assigned to them at every epoch. Epoch is defined differently from block because, as indicated above, there can be parallel blocks that can be produced by multiple nodes while epoch rewards will be constant per X seconds.

These rewards will be sent to the governing smart contracts for the nodes to claim during phase 1 and might change as an L1 comes into the picture.

The rewards are asymmetrical for all types of nodes, based on their cost and value to the network. The ratio for them can be changed through the governing smart contract in the future.

6.2.1.2 Fees from Transactions

Nodes also benefit from the fees that are paid while sending transactions to the network. The fees can be generated from Fees quota or via gracious native conversion as described in the white paper above.

However, in the current network design, the fees are not claimed by validators producing a block. Instead both the node rewards and transaction fees are shared among specific types of nodes equally though this might change as more simulations are done on the game theory involved.

6.2.2 Node Penalties

Nodes will be penalized for bad behavior to maintain the integrity and reliability of the network.

Penalties are enforced in two primary ways:

1. **Slashing**: A percentage X of the node's stake is deducted as a penalty.
2. **Banning**: A higher percentage Y of the node's stake is deducted, thus effectively removing the node from the network.

Note: The values of slashing, and banning are defined in the governing smart contract section and can be changed via governance.

Penalties are applied based on the following process:

- A node receives Z reports of bad behavior before being slashed.
- After accumulating a set number of slashes, the node is banned.

7. Network Security and Attack Analysis

A network's strength significantly depends on how effectively it handles malicious activity and bad actors.

7.1 Handling Malicious Validators

The Push Chain network has been designed to provide the utmost security against any malicious activity on the network.
We will analyze the case in which fresh malicious nodes can join the network by providing their staked tokens, assuming that they can interfere or stop the validation process.

In order to understand the probability of a malicious unverified transaction, let's define a few parameters and mathematically verify the security of the network:

- **V:** *Total number of validators*
- **VF:** *Validation Factor, Number of random validators picked for signing, i.e.,* ***10% of V*** *(by-default) but not less than 3.*
- **A:** *Number of validators controlled by an attacker.*
- ***a****: Minimum number of nodes an attacker needs to control for a successful attack*

Now, to successfully hack the network, the attacker needs to control at least 51% of the selected validators (VF). Keeping that in mind, let's perform the following calculations:

$$a = \lceil 0.51 \times VF \rceil$$

The probability(P) that the attacker controls at least a out of VF random validators is given by the sum of hypergeometric probabilities:

$$P(X \geq a) = \sum_{i=a}^{VF} \frac{\binom{A}{i}\binom{V-A}{VF-i}}{\binom{V}{VF}}$$

Example Probability Calculations

For these calculations, let's assume:

- V=20 (total number of validator nodes).
- VF (number of signatures required to validate a block) values ranging from 3, 5, 10, to 15.
- A (number of malicious validator nodes) ranging from 1 to VF

V	VF	A (value)	A (% of VF)	A (% of V)	a	Probability (%)
20	3	1	33.33	5	2	0.0000
20	3	2	66.67	10	2	1.5789
20	5	2	40.00	10	3	0.0000
20	5	3	60.00	15	3	0.8772
20	10	5	50.00	25	6	0.0000
20	10	6	60.00	30	6	0.5418
20	15	7	46.67	35	8	0.0000
20	15	8	53.33	40	8	5.1084

For safety let’s assume a safe probability that we might get a malicious block at <0.0001%.

Summary of Safe Values (for V=20 vnodes):|
For VF=15 , a safe amount of total malicious nodes is 7 (35%)
For VF=10 = 5 (25%)
For VF=5 = 2 (10%)

Now let’s try with a bigger network 10k/20k nodes with 32/64/128 verifications

V	VF	A (value)	A (% of VF)	A (% of V)	a	Probability (%)
10000	32	1490	-	14.90	17	0.0000
10000	32	1500	-	15.00	17	0.0001
20000	32	2960	-	14.80	17	0.0000

V	VF	A (value)	A (% of VF)	A (% of V)	a	Probability (%)
20000	32	2980	-	14.90	17	0.0001
10000	64	2280	-	22.80	33	0.0000
10000	64	2290	-	22.90	33	0.0001
20000	64	4560	-	22.80	33	0.0000
20000	64	4580	-	22.90	33	0.0001
20000	128	6000	-	30.00	66	0.0000
20000	128	6100	-	30.50	66	0.0001

Summary of Safe Values (for V=10k/20k nodes):
For VR=32, a 15% of malicious nodes (in a worst case scenario) will result only in 0.0001% chance of attack.
For VR=64 -> we can sustain 23%
For VR=128 -> we can sustain 30% bad actors

The table above demonstrates the probability that an attacker controls at least 51% of the selected validators under various scenarios.

1. **Higher Number of Validators:** When the total number of validators (M) increases, the difficulty for an attacker to control a majority of the selected validators also increases. This is because a larger pool of validators dilutes the influence of any single attacker.
2. **Higher Percentage of Validators Selected:** Increasing the percentage of validators selected (S) from the total validators (M) decreases the attacker's probability of controlling the majority. This is due to the higher number of validators required to reach a majority.
3. **Number of Compromised Validators:** The probability also depends on the number of validators (k) controlled by the attacker. With fewer compromised validators, the attacker's chance of controlling the majority diminishes significantly.

As we can see, **the probability of a successful attack significantly decreases as the number of validators (M) and the percentage of validators selected (S) increase.**

7.2 Handling Malicious Storage Nodes

In this section, we analyze the security of the sharding mechanism in the network to assess the probability of a successful attack by malicious nodes.

Parameters:

S: Total number of storage nodes.

A: Number of malicious storage nodes controlled by an attacker.

SH=32: Total number of shards in the network.

RF: Replication factor, representing the number of random nodes out of S that host a specific shard, defined by a smart contract algorithm.

A shard read is considered valid only if 51% of the nodes hosting the shard return the same value. An attack is considered successful if the attacker controls more than 51% of the nodes hosting a specific shard.

Minimum number of nodes an attacker needs to control for a successful attack

$$a = \lceil 0.51 \times RF \rceil$$

The probability P(X≥a) that the attacker controls at least a out of RF nodes is given by

$$P(X \geq a) = \sum_{i=a}^{RF} \frac{\binom{A}{i}\binom{S-A}{RF-i}}{\binom{S}{RF}}$$

Let's compute example probabilities for
S = 20 (number of storage nodes)
RF = 3..5..10..15 (replication factor, this is a number of copies of the same data shard across the network of storage nodes)

A = 1..RF (number of malicious storage nodes, we will start from 1 and go up to RF)

S	RF	A (value)	A (% of RF)	A (% of S)	a	Probability (%)
20	3	1	33.33%	5.00%	2	0.0000
20	3	2	66.67%	10.00%	2	1.5789
20	5	2	40.00%	10.00%	3	0.0000
20	5	3	60.00%	15.00%	3	0.8772
20	8	4	50.00%	20.00%	5	0.0000
20	8	5	62.50%	25.00%	5	0.3612
20	10	5	50.00%	25.00%	6	0.0000
20	10	6	60.00%	30.00%	6	0.5418
20	15	7	46.67%	35.00%	8	0.0000
20	15	8	53.33%	40.00%	8	5.1084
500	32	80	-	16.00%	17	0.0000
500	32	85	-	17.00%	17	0.0001
10000	32	1490	-	14.89%	17	0.0000
10000	32	1500	-	15.00%	17	0.0001
20000	32	2960	-	14.79%	17	0.0000
20000	32	2980	-	14.89%	17	0.0001
50000	32	7400	-	14.79%	17	0.0000
50000	32	7450	-	14.89%	17	0.0001

For safety,let's assume a safe probability that we might get corrupted data at <0.0001% for a data lookup for any shard.

Summary of Safe Values (for S=20 nodes)
For RF=15 , a safe amount of total malicious nodes is 7 (35%)
For RF=10 = 5 (25%)
For RF=8 = 4 (20%)
For RF=5 = 2 (10%)
For RF=3 = 1 (5%)

Summary of Safe Values (for S=0.5k,10k,20k,50k nodes)
For RF=32, a 15% of malicious nodes (in a worst case scenario) will result in only a 0.0001% chance of attack. In practice, randomized node assignment can significantly reduce the impact of malicious nodes, even if they constitute a higher percentage of the network than 15%.

The table above illustrates the probability that an attacker can control at least 51% of the storage nodes responsible for a shard, under various replication factors (RF) and numbers of malicious nodes (A). Here are the key points to understand:

1. Higher Replication Factor (RF):

When the replication factor (RF) increases, meaning more storage nodes are involved in hosting each shard, it becomes more difficult for an attacker to control a majority of those nodes. This is because the attacker needs to compromise a larger number of nodes to influence the shard data.

For example, with RF=15, an attacker would need to control 7 nodes (35%) out of 20 to reach a probability of <0.0001%, which is considered safe. This higher threshold makes it harder for an attacker to succeed.

2. Percentage of Malicious Nodes (A % of S):

The safe values indicate the maximum percentage of malicious nodes that can exist in the network without compromising shard security.

For instance, with RF=10, the network can tolerate up to 5 malicious nodes (25%) while maintaining a safe probability of a successful attack. This means that as long as the percentage of malicious nodes remains below this threshold, the probability of corrupted data remains extremely low.

3. Lower Replication Factor (RF):

When the replication factor is lower, fewer nodes are responsible for each shard, making it easier for an attacker to control a majority.

4. Impact of the Number of Malicious Nodes (A):

The probability of a successful attack significantly depends on the number of malicious nodes (A) controlled by the attacker. Fewer compromised nodes mean a lower chance of controlling the majority. As demonstrated, increasing the number of malicious nodes beyond the safe values greatly increases the probability of a successful attack.

As seen from the analysis, the probability of a successful attack decreases significantly as the replication factor (RF) and the number of storage nodes (S) increase. By maintaining a higher replication factor and ensuring the number of malicious nodes remains below the safe thresholds, the network can effectively safeguard against data corruption and maintain shard security.

This structured approach to shard security ensures that **the probability of obtaining corrupted data remains low**, providing a robust defense against potential attacks.

8. Phase 2: Seamless Interoperability

While the initial phases of the roadmap primarily solve for scalable consumer centric applications, the next phase aims at enabling seamless interoperability between chains that enables movement of value or message payload to support cross chain features in the apps.

This phase leverages a solver network capable of instant cross-chain transactions to ensure better and faster UX for cross chain apps.

This is an intermediate requirement to enable a chain that provides seamless interoperability and hyper-scalability from Phase 1. This further opens up the possibility of building scalable universal apps that aren't tied to one specific chain, are scalable, and provide better UX without forcing app users to deal with any on-chain intricacies

As Phase 2 rolls out, we aim to introduce some crucial modifications to the already existing network:

1. **The introduction of a solver network:** We plan to implement a solver network to enable faster interop between chains. Unlike traditional interoperability, this won't rely on the finality of chains and instead leverage specialized on-chain actors to perform the interoperability requirement of the user or apps built on top of the chain.

2. **The activation of arbitrary message bridging** along with value transfer is a major plan for this phase. Arbitrary message bridging (a.k.a AMB) is a mechanism to bridge any arbitrary message payload between chains.

   Irrespective of the type of the AMB, this simple feature shall open the doors for a wide range of cross-chain web3 applications that require faster scalable chains for cross-chain communication.

3. **The enabling of strict and flexible ordering of arbitrary messages** is another modification that comes along with the arbitrary message feature.

   a. **Strict ordering** here refers to the fact that a batch of arbitrary messages from an application must be processed in a strict order, i.e., adhere to the

nonce. This is for serious messages where processing of messages in a strict order is a compulsion. While this could be comparatively slower, it does achieve the secured processing that it demands.

b. **Flexible ordering,** on the other hand, refers to batch or arbitrary messages that don't need to be processed in a strict order. These are messages that can be processed faster, quite similar to consumer transactions, as they aren't bound by any strict order.

The strictness and flexibility in message ordering provides an additional layer of flexibility for developers to create AMB-specific cross chain applications that might or might not require strict ordering, thus allowing them to easily choose between security vs speed based on requirements.

4. **The introduction of Relayers / Solver nodes** will be a new type of actor/operator on the Push Network introduced from Phase 2 onwards. The relayer nodes' primary goal is to ensure the safe and secure delivery of cross chain messages/values from source to destination gateway smart contracts (a.k.a, *Push smart contracts deployed on multiple chains that act as entry points for initiating any cross-chain requests)*.

9. Phase 3: Unified L1 and Universal Apps

The final phase, i.e., Phase 3, is the most crucial one as it combines all the phases together to create a complete L1 with arbitrary state transition capabilities that supports the creation of universal smart contracts on top of Push Chain.

Considering the strong emphasis on consumer-centric applications as well as on the unification of the multi-chain landscape, this chain will foster faster, cheaper and seamlessly interoperable applications for web3.

Some crucial roadmap items for this phase involves:

1. **The enabling of arbitrary state transitions** in the Push Chain. Phase 1 included stateless blocks for faster processing of consumer data and Phase 2 ensured the transfer of monetary value as well as arbitrary message bridging in an ordered or unordered fashion. Phase 3, however, leads to a much more extensive solution by providing a L1 that is capable of arbitrary state transitions, i.e., running its own unified smart contracts.

2. **The Enabling of Unified Storage** which means that the L1 not only has read and write storage to its own state but is also able to provide read-only access to all the other chains that Push will support along with read access to consumer apps and monetary app transactions.

3. **The Enabling of an abstracted future for web3 users** where users shall no longer be required to keep track of multiple wallets for N number of chains. Instead they can use an abstracted wallet that can be used for any on-chain activity, keeping track of multi-chain funds of a specific user, without having to switch between chains or wallets.

10. Conclusion

In this paper, we presented the vision and implementation of Push Chain. Push Chain aims to expand the boundaries for web3 by creating a shared state layer 1 chain that prioritizes the support and development of universal and hyper-scalable consumer-centric apps in web3 while ensuring fast finality, cheap transactions, and cost-effective storage as well as apps with shared state experience.

This network addresses the specific needs of modern decentralized consumer applications that are capable of paving the way for the mass adoption of Web3. Additionally, it aims to resolve the broken UX and fragmented ecosystem of web3 by creating a network that enables the development of Universal applications that allow users to seamlessly continue on-chain activity in an abstract fashion where they never have to worry about the underlying technology behind the applications.

Push Chain enables a Proof-of-Stake (PoS) blockchain network that introduces a new set of mechanisms like *shared state L1, parallel block execution, dynamic sharding, fee abstraction and wallet abstraction* to enable the development of hyper-scalable applications. These applications have read-and-write capabilities that scale with the growth of the number of nodes in the network, providing an infrastructure that can handle the needs of thousands of applications for speed and scalability.

The paper divides the entire roadmap of building this decentralized network into 3 different phases. Phase 1 mainly involves creating a PoS network that allows consumer transactions and enables the rapid development of consumer-centric applications in web3. Phase 2 introduces seamless interoperability between chains for users and apps by leveraging a solver network. And, Phase 3 is aimed at bringing the era of universal apps with shared state feature. With Phase 3, the chain aims to create a universal and unified experience where developers can create applications without being limited to a specific chain or having to migrate between chains and users are able to navigate through web3 apps without ever having to learn about underlying chains and native gas tokens, thus enabling a traditional-web like experience.

Together, all these phases lead to a network that is capable of fostering the creation of hyper-scalable and universal consumer apps that lead to an abstracted future of blockchain networks. This reduces the complexity for developers to build the next

generation of interoperable & chain-agnostic applications as well as for users to experience a chain that is truly abstracted with enhanced UX for web3 apps.

Disclaimer

PLEASE READ THIS SECTION CAREFULLY. IF YOU ARE IN ANY DOUBT AS TO THE ACTION YOU SHOULD TAKE, YOU SHOULD CONSULT YOUR LEGAL, FINANCIAL, TAX, OR OTHER PROFESSIONAL ADVISOR(S).

Readers are advised to conduct their own independent research and consult with professionals in the relevant fields before making any decisions related to Push Chain or any associated projects. We expressly disclaim any and all responsibility for any direct or consequential loss or damage of any kind whatsoever arising directly or indirectly from reliance on any information contained in the whitepaper, as well as from any error, omission or inaccuracy in any such information or any action resulting therefrom.

1. No Guarantee: We do not guarantee the accuracy of or the conclusions reached in this whitepaper, and this whitepaper is provided “as is.” This whitepaper does not make and expressly disclaims all representations and warranties, express, implied, statutory or otherwise, whatsoever, including, but not limited to: (i) warranties of merchantability, fitness for a particular purpose, suitability, usage, title or non-infringement; (ii) that the contents of this whitepaper are free from error; and (iii) that such contents will not infringe upon third-party rights.

2. Future Changes and Developments:. The information contained in this whitepaper might become outdated or may not reflect the current state of the Push Chain platform or its future iterations. The Push Chain development team reserves the right to make changes or improvements to the platform without prior notice.

3. Regulatory Uncertainty: We operate in an emerging and disruptive industry that is under rapid and dynamic development. We are subject to various laws, and are at risk in relation to changes in the laws and the timing and effects of changes in the respective legislation. This includes changes in the interpretation thereof which cannot be predicted by us.

4. Potential Risks: Engaging with Push Chain, like all blockchain and cryptographic projects, carries inherent risks. This includes but is not limited to technological vulnerabilities, market volatility, cyber-attacks, or other unauthorized use of data, other malicious interference and third-party actions or interventions. It is imperative that potential users and participants understand these risks before engaging with Push Chain.

5. Investment Warning: This whitepaper should not be considered as a solicitation or offer to buy or sell any security or as a recommendation to invest in Push Chain. Potential investors should be aware that investments in cryptocurrencies can be high-risk and past performance is not indicative of future results.

6. Liabilities: The Push Chain development team, contributors and any associated parties shall not be liable for any losses or damages, whether direct or indirect, arising from the use of or reliance on the information contained in this whitepaper. Anyone considering using Push Chain or engaging in associated activities should conduct thorough due diligence and seek advice from appropriate professionals.