WalletConnect Network: A high throughput and low latency globally distributed network

The future of permissionless onchain UX

Daniel Martinez
daniel@walletconnect.com

Derek Rein
derek@walletconnect.com

Ivan Reshetnikov
ivan@walletconnect.com

Nodar Darksome
nodar@walletconnect.com

Rafael Quintero
rafael@walletconnect.com

Special thanks to Pedro Gomes, Wei Dai, Robert Koschig, and Timofey Yaluhin for their feedback and review.

Abstract

WalletConnect is a cornerstone of the web3 industry supporting over 20 million monthly dapp to wallet connections. Connections are facilitated by the *Relay* service, which is a stateful application using the *WalletConnect Network* as its storage backend.

The *WalletConnect Network* is a globally distributed set of nodes, each housing a portion of the dataset that backs the *Relay* service and other applications that operate on top of the Network. These nodes persist data while automatically managing expiration and cleanup based on configurable parameters.

The Network is currently operated in a permissioned model by over 16 distinct entities running WalletConnect Nodes. It processes ~70,000 requests per second at ~180ms p50 latency.

This paper discusses the technical implementation of the Network, the economic incentives, the fully permissionless future, and as such its potential as a new primitive in the decentralized application builder's toolbox.

1. Motivation

The WalletConnect Network represents an advancement in distributed systems and database technology as it is a key-value store, operated not only by one but currently 16 distinct entities. As of January 2025, the Network processes an average of ~70,000 requests per second with low latency, supporting over 20 million monthly connections between decentralized apps and wallets. This level of performance positions it as a potential new primitive in the decentralized app and decentralized application builder's toolbox: a high-throughput, low-latency key-value store.

Evolution

WalletConnect launched in an environment where there was a growing number of mobile Ethereum wallets trying to support their users to connect to desktop apps. Instead of each wallet having to craft its solution and apps having to implement each wallet's solution, WalletConnect provided a unified, end-to-end encrypted way for apps and wallets to communicate with one another. By Mid-2023 WalletConnect v1 was sunset, in favor of WalletConnect v2, which provided a multi-chain interface instead of EVM specific application layer and a multiplexed, more robust, IO-efficient, and scalable transport layer as well as enhanced security features. Most importantly, it laid the groundwork for what would become the WalletConnect Network - a decentralized storage layer powering the protocol's core services.

The Need for Decentralization

The impetus for developing the WalletConnect Network emerged from two primary sources. First, end-users demand resilient infrastructure that aligns with web3's ethos of decentralization. Second, dapps and wallets seek to reduce their dependency on centralized services that could become single points of failure. This highlighted the need for a decentralized solution that could maintain the high-performance requirements of the protocol.

Technical Challenges and Innovation

When designing the Network, no existing off-the-shelf decentralized system met the stringent requirements for latency and throughput needed to support WalletConnect's global user base. Traditional blockchain-based solutions by design suffer from network and latency overhead and the need for full replication, while distributed hash tables like Kademlia sacrifice latency for decentralization. This gap in the technology stack led to the development of a novel architecture based on rendezvous hashing.

Current Implementation and Future Applications

While the Relay service remains the primary application running on the Network, it is designed to and has proven versatile enough to support additional use cases. For instance, Smart Sessions now leverages the Network to store delegation policies, demonstrating its potential as a general-purpose decentralized and soon permissionless database.

A New Primitive for Decentralized Applications

As the Network transitions towards full permissionlessness, it has the potential to become a fundamental building block for decentralized applications beyond WalletConnect itself. Its ability to provide consistent low latency at scale, while maintaining the properties of decentralization, positions it as a potential solution for other protocols and applications requiring high-performance, decentralized, and comparably cheap key-value storage.

The following sections detail the technical architecture, consensus mechanisms, and incentive structures that enable the Network to achieve these capabilities while maintaining its security and decentralization properties.

2. Problem

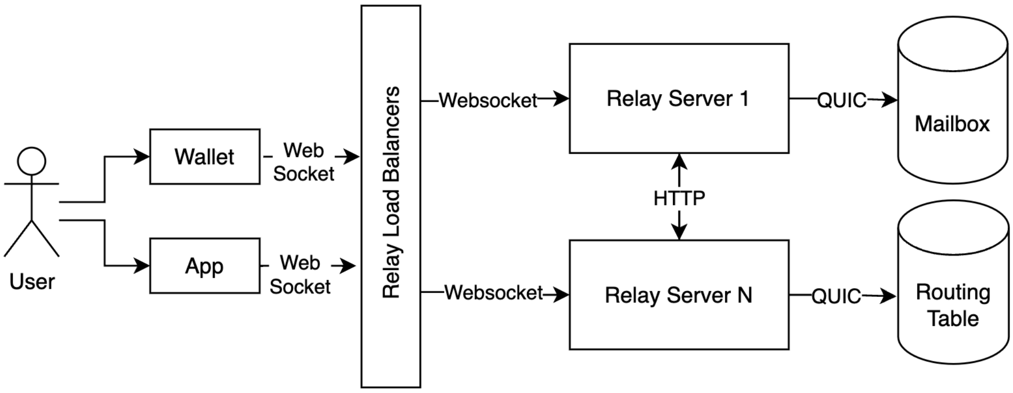

Figure 1: High-Level Architecture of the Relay

Relay Architecture
The WalletConnect Relay Service [1][2][3], is the main application on top of the WalletConnect Network, it is a stateful application that allows apps and wallets to communicate with one another over an end-to-end encrypted channel. Initially, apps and wallets were usually on different devices namely a desktop app and a mobile wallet. However, over time apps and wallets have increasingly operated on the same mobile device. Furthermore, without a stateless way of communicating with one another due to operating system constraints.

Operating systems commonly terminate connections when apps go into the background. Hence, stateful websocket connections cannot satisfy the requirement of apps/wallets going into the background. To satisfy this requirement the system uses a *Mailbox* for apps/wallets to buffer information until they come back online. To allow apps/wallets to connect to different websocket servers, sometimes in different regions, a *Routing Table* helps route messages to the right connected clients.

Low Latency
Is a core requirement decentralizing the system needed to fulfill as existing decentralized applications usually trade off one of the following: scale, worst-case latency, and cost. WalletConnect Relay required a solution that doesn't trade off any of these. Ultimately latency of the decentralized solution needed to be comparable with an existing off-the-shelf solution.

3. Prior Art

There have been a variety of innovative approaches to building decentralized or distributed storage and databases. Each system has contributed valuable concepts, ranging from replication strategies and consensus mechanisms to incentive models, and several served as direct inspiration for the WalletConnect Network. Below, is a brief outline of a few of these systems and highlights how their design goals and trade-offs differ from those of the WalletConnect Network.

Consistent/Rendezvous Hashing Databases
Popular distributed databases like Cassandra, DynamoDB, and MongoDB use consistent or rendezvous hashing to balance data across clusters, achieving impressive scalability and low latency. However, they typically rely on a centralized or permissioned operational model. While these databases provided key insights into shard allocation and replication strategies, they are not inherently designed with Byzantine-Fault tolerance in mind, a critical component for permissionless and decentralized architectures.

Blockchains

Public blockchains e.g. Ethereum are the OG permissionless systems. They showcase robust peer-to-peer networking, consensus, and immutable ledger properties. However, the overhead of global broadcast, consensus, and full-state replication means they prioritize security and transparency over millisecond-level latency and high throughput. The WalletConnect Network draws on blockchain principles for decentralization and trustlessness but requires a different performance profile.

Distributed Hash Tables

Protocols like Kademlia demonstrate permissionless, peer-to-peer lookups across a large network of participating nodes. Kademlia's design strongly influenced the peer discovery mechanisms in many decentralized systems. Yet, its unstructured approach and hop-by-hop routing trade-off are consistent, low-latency guarantees - making it less suited for globally distributed, near-real-time requirements.

Sharded Blockchains

Sharding techniques improve blockchain scalability by distributing data and transaction processing across multiple "shards." This helps increase throughput while retaining decentralized security. However, these systems still rely on a ledger-based model and full consensus at the shard level, often resulting in higher commit latencies that don't meet the requirements.

Data Availability Solutions

Systems like EigenDA or Celestia focus on ensuring transaction or rollup data is publicly retrievable, enabling off-chain computation with on-chain security guarantees. Though they excel at providing verifiable data for extended periods, they are optimized around immutability and verification rather than near-instant reads and writes. Their cost and finality models can be prohibitive for real-time, high-volume use cases.

Decentralized File Storage

Solutions like Filecoin and Arweave offer decentralized, censorship-resistant storage for large files, often with a focus on long-term or permanent availability. While they showcase effective incentive structures and distributed proofs of storage, their architectures are optimized for throughput and cost models tailored to larger, less frequently accessed data.

Other Domain-Specific Approaches

Projects like Space & Time or OpenDB have introduced domain-specific features - such as advanced query languages or OLAP-style processing - to meet specialized needs. Although these systems are valuable for their respective use cases, they typically involve heavier data replication or more complex querying than the WalletConnect Network, which emphasizes high performance with minimal overhead.

Taken together, these projects have paved the way for decentralized systems with varying performance characteristics, consensus strategies, and economic models. The WalletConnect Network draws upon insights from many of these efforts, combining elements of rendezvous hashing, peer-to-peer networking, and cryptographic security to offer a high-throughput, low-latency, and eventually permissionless database that may become a new primitive in the decentralized application builder's toolbox.

4. Specification

4.1. Node Protocol Overview

A node within the WalletConnect Network is a machine that runs the wcn-node software to provision storage capacity to the network. This capacity is leveraged by various clients—primarily the Relay service, Smart Sessions Service, and others in development—that rely on the network as a critical piece of infrastructure for their operation.

Nodes provide an API for performing various key-value store operations as Remote Procedure Calls (RPCs) using a custom framework built on the QUIC protocol. They use RocksDB as a storage back-end, with different column families dedicated to storing distinct data types. (see 4.3.2.).

Nodes, also known as replicas, are organized into replica sets, each responsible for storing multiple portions of the key space—referred to as shards—that contain the key-value pairs clients read and write (see 4.4.).

Currently, nodes join the network through a whitelisting process managed by our team via bootstrap nodes. In the future, when the system is permissionless, the whitelisting requirement will be lifted (see 6.3.).

The network uses Raft [4] to maintain a shared state, —a critical component for both authorization (see 4.5.) and data replication (see 4.4.). This consensus mechanism is provided as an API that nodes use to replicate the shared state and to communicate the addition or removal of peers.

Additionally, nodes use cryptographic primitives for identification, for secure node-to-node and client-to-node communication (see 4.6.1.) and to ensure data isolation (see 4.6.3.).

To maintain network quality and reliability, nodes also participate in the network's economic layer through a dual incentive mechanism combining staking and performance-based rewards (see 5.). Node operators must maintain a minimum StakeWeight by locking WCT tokens and consistently meet performance metrics to receive rewards. This economic model ensures high-quality service provision while aligning operator incentives with network reliability and performance goals.

4.2. Network Architecture

The network is designed to operate with globally distributed node deployments, each ensuring data availability in Europe, Asia, and North America. Nodes are also labeled according to their organization, enabling the distribution algorithm (see 4.4.) to ensure that every entry is replicated globally.

At the time of writing WalletConnect Foundation operates 8 nodes, with another 16 nodes operated by partners. This distribution is subject to change in the future as the network's community grows and more participants join.

The network employs a node prioritization strategy to maximize availability. Currently, this strategy prioritizes nodes controlled by us for read/write operations before considering third-party nodes. As the community of node operators grows, third-party nodes will gradually replace those the foundation controls.

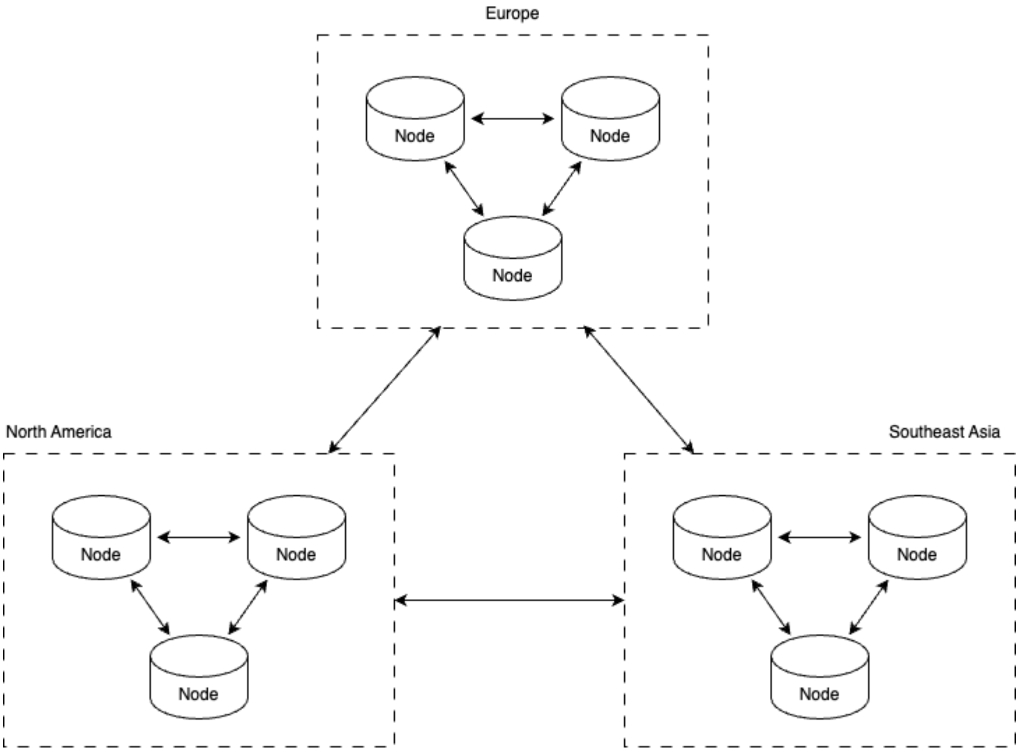

Figure 2: High-level overview of the geographical distribution of the WalletConnect Network.

4.3. Node API

Clients that interact with the network primarily interact with the Authentication API and the Storage API.

4.3.1. Authentication API

The purpose of this API is for clients to authenticate and obtain an authorization token to perform storage operations. Nodes maintain a whitelist of client Peer IDs [6] that are authorized to perform storage operations. Connecting client's Peer ID is derived from its TLS certificate during connection handshake (see 4.6.1.), and is then checked against the whitelist of allowed clients. If the client is on the list, a security token is issued for that client that is then used as authorization for the storage API. Each node can maintain its whitelist of allowed clients, and they are not synchronized among the nodes in the cluster. This means that the clients should have a list of known nodes to authenticate against.

The authorization token contains information about the issuing node (its Peer ID), its purpose (e.g. storage API), and a list of private namespaces the client is authorized to access (see 4.6.3.). Any node in the cluster can issue authorization tokens.

Another function of this API is to fetch cluster state and monitor changes in the cluster (for example, nodes leaving or joining the cluster). The replication and consistency logic lives entirely in the client, so it must have up-to-date information on the entire cluster at all times.

4.3.2. Storage API

Once the client has been authenticated and authorized to perform storage operations, it will use the token to open connections directly to nodes to perform storage operations. Upon opening a connection to the storage API, the client presents an authorization token. The token issuer's Peer ID is then checked against a known node list, and if the Peer ID is one of the nodes in the cluster, the client may proceed with performing storage operations.

Each storage operation requires a key, which is composed client-side and then hashed to determine the replica set responsible for storing this key. The client connects directly to each node in the replica set to perform storage operations on that node. All of the data modification operations also require a version, which is used for conflict resolution (see 4.4.2.).

Storage operation is performed concurrently on all nodes in the replica set and is considered successful as soon as the majority of nodes have acknowledged the operation.

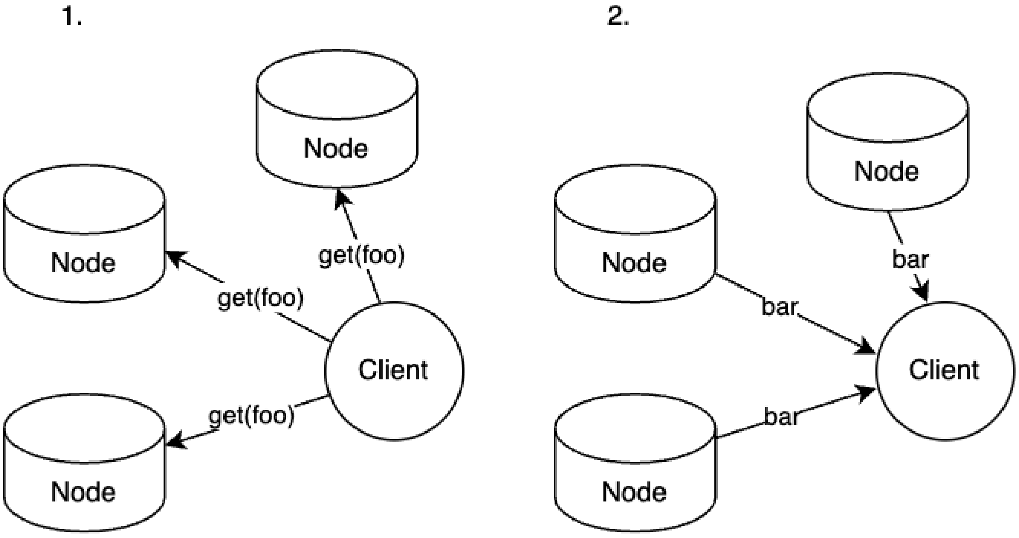

Figure 3: Successful read operation

RPC	Arguments	Description
get	key	Returns the value of a key.
set	- key - value - expiration - version	Sets the value of a key.
del	- key - version	Deletes the value of a key.
get_exp	key	Returns the expiration time of a key.
set_exp	- key - expiration	Sets the expiration time of a key.

	- version	
hget	- key - field	Returns the value of a field in a map.
hset	- key - field - value - expiration - version	Sets the value of a field in a map.
hdel	- key - field - version	Deletes a field and its value from a map.
hget_exp	- key - field	Returns the expiration time of a field in a map.
hset_exp	- key - field - expiration - version	Sets the expiration time of a field in a map.
hcard	key	Returns the number of fields in a map.
hscan	- key - count - cursor	Returns entries of a map by iterating over fields and values.

4.4. Replication Strategy

Data storage and replication in the network are achieved using a variant of Consistent Hashing called Rendezvous Hashing [5], and in addition to that, a heuristic to select nodes based on certain attributes each node can have based on each node's region and organization configuration.

Our implementation of Rendezvous Hashing first distributes keys evenly between, roughly, 65,000 shards and then assigns replica sets to shards. Each replica set is currently configured to consist of 3 nodes, selected based on their organization and region (see 4.5.). This policy ensures nodes are from different organizations and hosted at different regions, thus making the network resilient to region-wide outages.

With the key space split into shards, and shards having assigned replica sets, finding nodes responsible for storing a key uses the following algorithm:

- Hash the key into an 8-byte integer;
- Use the first 2 bytes of the hash as the shard identifier;
- Lookup the replica set for that shard using a precomputed lookup table.

Once the client has obtained a replica set for the key, it proceeds to perform a storage operation on each node in the replica set individually (see 4.3.2.).

4.4.1. Key Space Redistribution

As nodes join and leave the cluster, the key space must be updated and rebalanced. As a result of such an update, some nodes may be assigned to different shards.

The nodes joining the cluster are required to fully download the data for shards they've been assigned to, while the leaving nodes are required to transfer the data to the nodes replacing them on the key space.

In this context, Rendezvous hashing helps minimize the number of nodes affected by a key space redistribution.

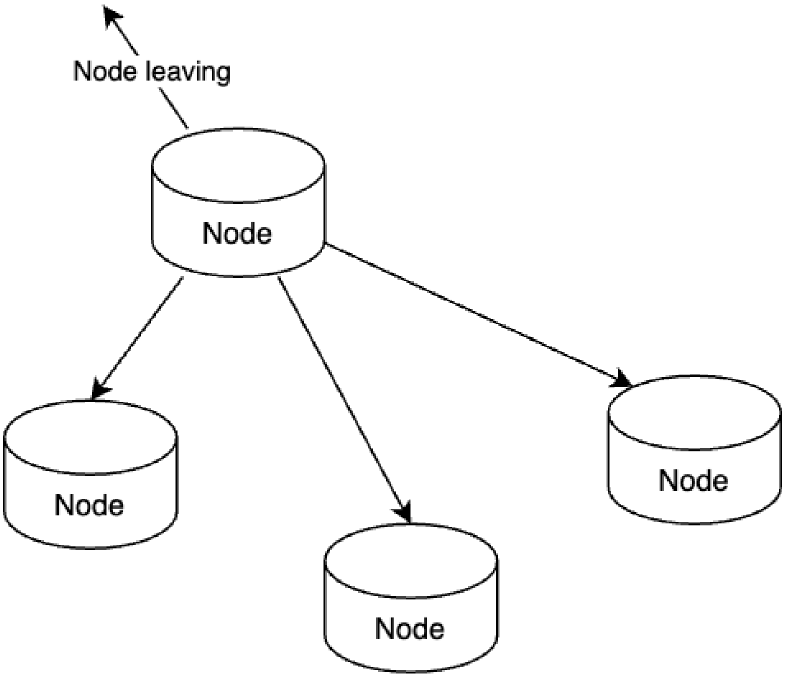

Figure 4: Rebalancing occurs when a node attempts to leave the network.

4.4.2. Conflict Resolution

Conflicts can occur when two client instances are attempting to update the same value concurrently.

Nodes apply a last-write-wins conflict resolution strategy, meaning that the most recent version of a value, based on its timestamp, is considered the correct one, and previous versions of the value are discarded.

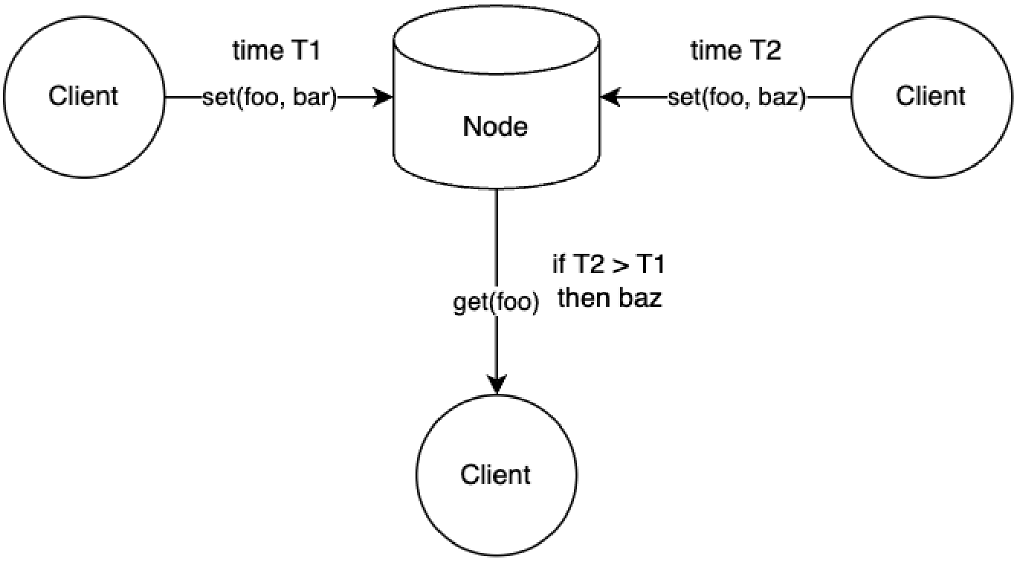

Figure 5: How to write conflicts get resolved by applying the last-write-wins strategy.

4.4.3. Read-Repair Mechanism

Read repair is a mechanism used to correct inconsistencies among replicas during a read operation. When a client requests data, it compares responses from multiple replicas. If one or more replicas return outdated or inconsistent data, the system automatically updates those replicas with the correct value, helping to maintain data consistency across the network over time.

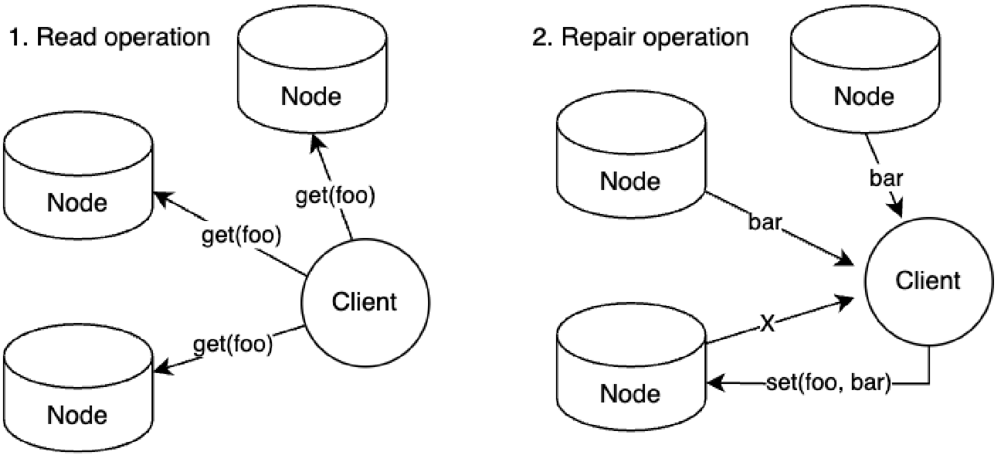

Figure 6: Read repair process.

4.5. Replica Consensus

In the node Protocol, Raft maintains a data structure called the Cluster View. This shared structure contains information about every node in the network, including each node's Peer ID, IP address, geographical service region, and affiliated organization, along with other protocol-relevant details.

Another function of Cluster View is maintaining a precomputed key space lookup table, used by clients to perform storage operations (see 4.4.), as well as ongoing data migration status (see 4.4.1.).

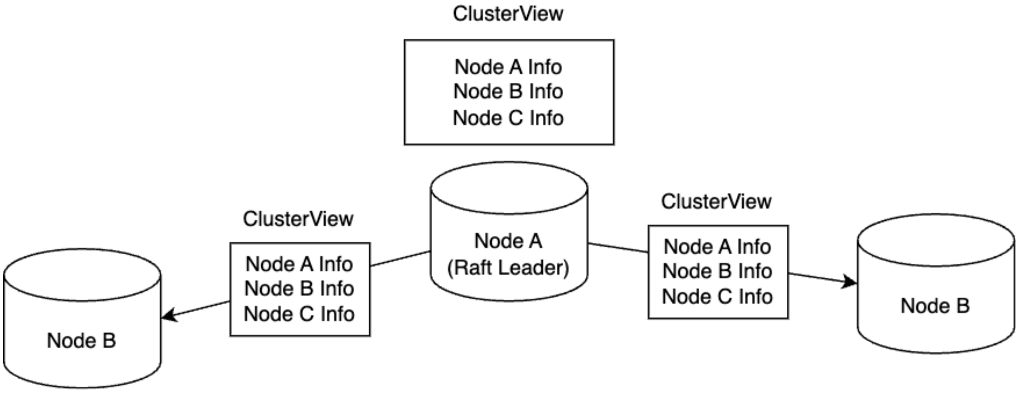

Figure 7: Consensus ClusterView replication

4.6. Encryption

4.6.1. Encryption in Transit

All communication between nodes and the client API is secured by TLS 1.3 [7] encryption. The certificate authentication process is based on the libp2p TLS handshake [8] protocol. Certificates generated this way contain cryptographic proof that the peer was in possession of the private key from which its Peer ID is derived. Hence any server-to-server or client-to-server connection requires a known Peer ID so that it can be verified during the handshake to prevent a possible man-in-the-middle attack.

4.6.2. Encryption at Rest

Nodes provide no encryption of data at rest, and it's up to the client to decide how and when to encrypt the data stored in WCN. As an example, dapp/wallet clients using WalletConnect Protocol use the Relay and Network as a communication channel to establish an end-to-end encrypted channel between each other.

4.6.3. Storage Namespaces

With all of the data in WCN being internally stored in shared column families, care must be taken when considering storage key structure. By default, all data stored is accessible to all clients without any constraints. Anyone knowing the storage key can fetch, modify, or delete data stored at that key.

To secure their data and prevent key collisions, clients may opt to use a private storage namespace. Accessing data in a private namespace is possible only after completing a challenge-response authentication using the Authentication API (see 4.3.1.). The process of authentication requires the client to have an Ed25519 keypair. The public key of the key pair is used to prefix all storage keys in the namespace, effectively separating those keys from the global data. The private key is needed to complete the authentication by signing a server-provided nonce.

By default, private data is encrypted and signed by the client using the AEAD method with ChaCha20-Poly1305 cipher to prevent possible corruption or tampering by the node operator.

Data encryption is not a requirement of the Storage API and can be implemented differently by the client or omitted. The only requirement for accessing a private namespace is an Ed25519 key pair.

4.7. Network

As mentioned in the replication strategy section (see 4.4.), each storage operation makes 3 requests (one per deployment region) and requires at least 2 successful responses.

Following is the average request latency under normal conditions as observed from different client deployment regions:

- Europe: 96ms.
- North America: 96ms.
- Asia-Pacific: 178ms.

The above numbers are comprised primarily of round-trip network latency, as the locally executed storage operations (performed by the RocksDB backend) all have <1ms latency, including collection iteration (i.e. hscan).

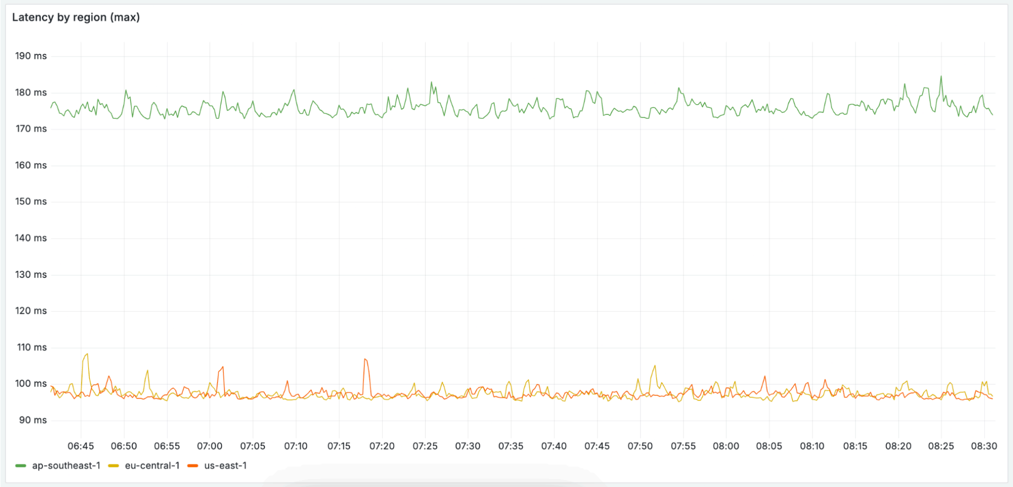

Figure 8: Grafana chart displaying max latency by region.

4.7.1. Transport Layer

The transport layer protocol used for both server-to-server and client-to-server communication is QUIC [9]. The decision to use QUIC over TCP was guided mainly by high-request concurrency and throughput requirements. Currently, the network performs anywhere between 40,000 and 70,000 storage operations per second.

Internally, the RPCs used for all WCN communication rely on stream multiplexing, meaning a separate stream is opened for each RPC request. This type of multiplexing allows for reduced head-of-line blocking compared to alternatives (especially over TCP), which results in higher overall throughput and improved latency.

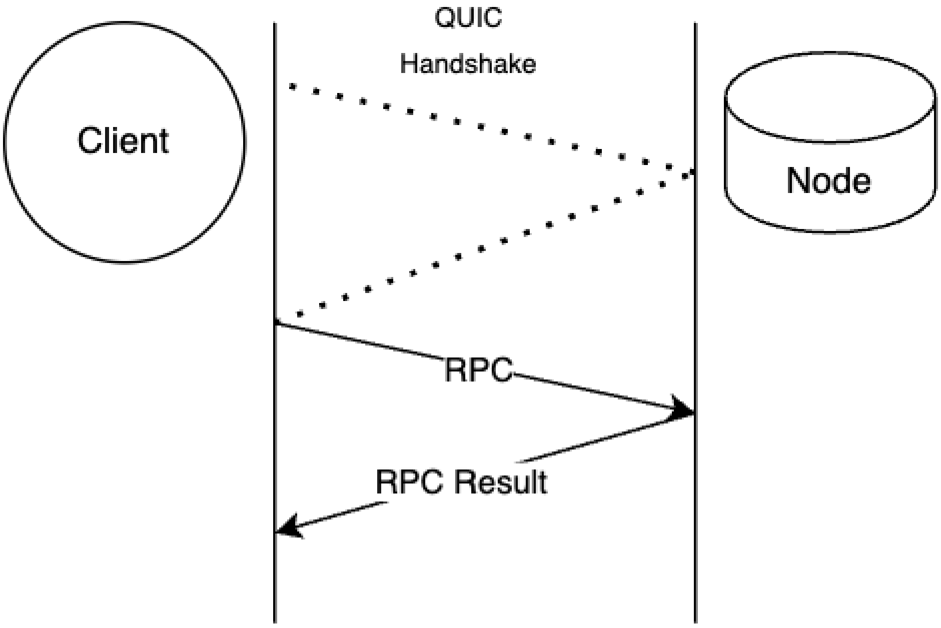

Figure 9: Flow of RPCs over QUIC.

5. Incentive Layer

The WalletConnect Network implements a performance-based reward system that incentivizes node operators to maintain high availability and optimal service quality. This section details the economic mechanisms that govern node operator rewards, performance measurement, and network participation.

5.1. System Architecture

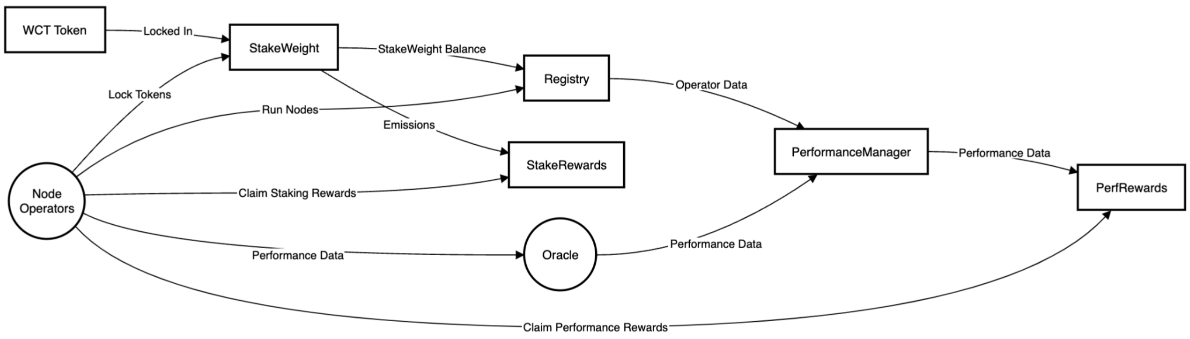

Figure 10: Incentive Layer Architecture

The WalletConnect Network's economic system consists of five core components: The WCT Token and StakeWeight component serves as the foundation, providing the network's native token for staking, managing token locking with decay-based voting power, and controlling staking reward emissions.

The Operator Registry handles node registration and authentication processes. Performance Management consists of three key elements: an oracle collecting raw performance data, a PerformanceManager validating and processing metrics, and a distribution system that feeds validated data to rewards and tracks operator eligibility.

The Reward Distribution system operates through two main mechanisms:

- **NodePerformanceRewards:** handles operator's performance-based rewards and StakeWeight verification
- **StakingRewardDistributor:** manages general staking emissions and processes reward claims.

This architecture ensures a clear separation between performance measurement, reward calculation, and safety controls while maintaining the connection between staking and operational requirements.

5.2. Node Identity and Authentication

Nodes in the network maintain two distinct identities:

1. An Ed25519 keypair for network operations and message signing
2. An OP Mainnet address for receiving rewards and participating in governance

The system provides two key management operations:

5.2.1. Key Rotation

Node operators can rotate their Ed25519 keys while maintaining their network identity through a secure signature-based process. This enables routine security maintenance without disrupting network operations. The rotation process requires cryptographic proof of ownership of both the current and new keys.

5.2.2. Operator Transfer

The system supports the full transfer of node ownership between operators. This process requires cryptographic authorization from both the current operator (using their Ed25519 key) and the new operator (using their Ethereum wallet), ensuring secure and verifiable ownership transitions.

5.3. Epoch Structure and Time Management

The network operates in weekly epochs aligned with Unix timestamps, with each epoch starting at 00:00 UTC on Thursdays. This alignment ensures deterministic epoch calculations in Solidity smart contracts, which operate on Unix time. The Thursday boundary provides a predictable schedule for operators while avoiding weekend transitions.

As illustrated below, each epoch follows a cyclical flow through distinct phases:

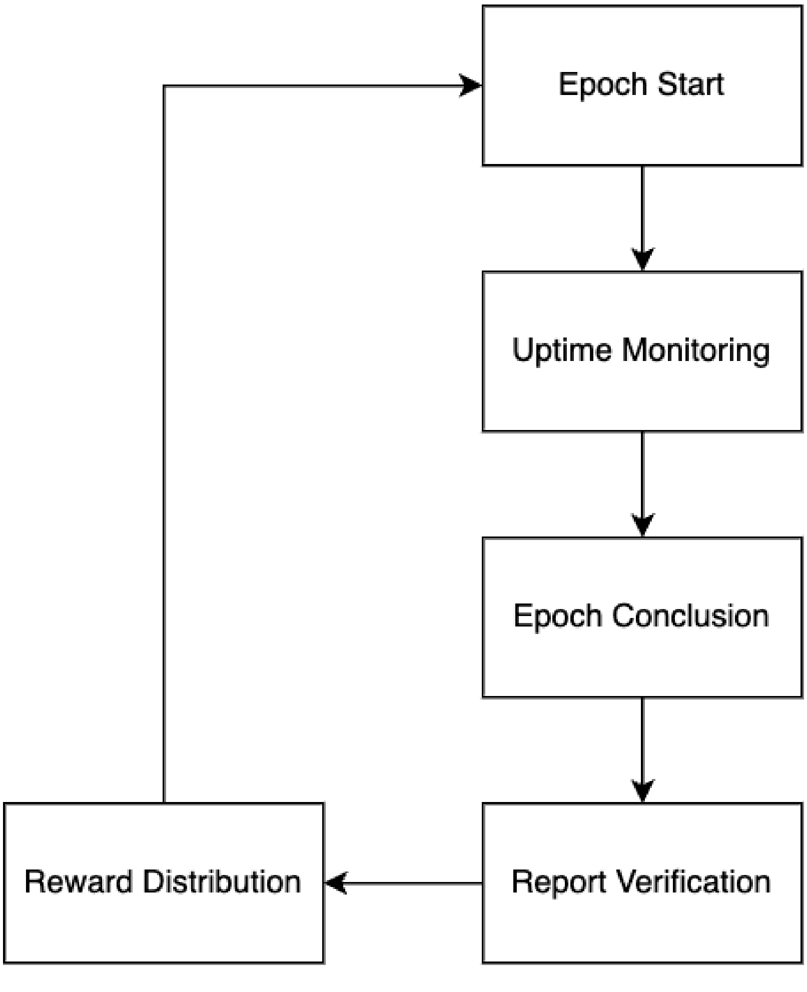

Figure 11: Node Rewards Epoch Lifecycle

The process begins with Epoch Start, where StakeWeight requirements are verified and performance tracking is initiated. During the week-long Operational Period, the network continuously monitors uptime and collects performance metrics.

At the Epoch Conclusion, final metrics are collected and the cycle transitions into a 24-hour Finalization Period. This crucial period allows for thorough data verification and report preparation. Finally, the Distribution Phase enables operators to claim rewards based on their performance, while the next epoch is already underway.

This overlapping structure ensures continuous network operation while maintaining rigorous validation and transparent reward distribution. Each phase serves a specific purpose in maintaining network integrity and operator incentives.

5.4. Access Control and Permissions

The system implements role-based access control (RBAC) to manage different system functionalities:

- **Operators**: Can participate in the network and claim performance rewards
- **Oracle Role**: Authorized to submit operators' performance metrics
- **Admin Role**: Can manage system parameters and emergency functions
- **Registry Manager**: Controls node registration and removal

During the initial phase, these roles are managed by the WalletConnect Foundation.

5.5. Performance Measurement

5.5.1. Performance Oracle

A dedicated Oracle service aggregates performance data across the network and submits it to the reward distribution system. In the current permissioned phase, the network operates under trust assumptions regarding operator-reported metrics, as the system is designed for participants who are vetted and expected to operate in good faith. This approach aligns with the network's initial focus on establishing operational stability with trusted partners.

5.5.2. Performance Metrics

The system is designed to track two primary performance indicators: uptime and latency.

5.5.2.1. Uptime Scoring

Each node's uptime score (USi) is calculated based on its absolute availability:

$$\mathrm{U}ptimeScore_i = \frac{Uptime\ Units(i)}{T}$$

Where:

- T is the total number of time units in the epoch (e.g., the number of 15-second intervals in a week)
- $UptimeScore_i$ = 1 represents 100% uptime

Performance scores are calculated with a precision factor of 10,000 (basis points) to handle decimal calculations in Solidity. For example, an uptime of 99.5% would be represented as 9950 basis points.

5.5.2.2. Latency Scoring

While not currently implemented, future implementations will incorporate node-specific latency measurements, scored on a scale of 0 to 1 using the following formula:

$$LatencyScore_i = \max\left(0, \frac{L_{max} - L_i}{L_{max} - L_{min}}\right)$$

Where:

- L_i is the node's average latency
- $Lmax$ is the maximum acceptable latency
- $Lmin$ is the theoretical minimum latency
- $LatencyScore_i$ ranges from 0 (worst) to 1 (best)

5.5.2.3. Combined Performance Score

The system combines these metrics into a weighted performance score:

$$P_i = w_1 \cdot UptimeScore_i + w_2 \cdot LatencyScore_i$$

Where:

- w_1 and w_2 are configurable weights ($w_1 + w_2 = 1$)
- Initially, $w_1 = 1$ and $w_2 = 0$ (uptime-only scoring)
- All scores maintain 10,000 basis points precision

5.5.3. Measurement Parameters

- Performance data is collected and aggregated in 1h intervals
- Weekly epochs align with Unix timestamps (Thursday 00:00 UTC)
- 24-hour finalization delay ensures data accuracy
- Reports include total uptime units and derived performance scores

The Performance Manager, the Smart Contract responsible for the introduction of additional metrics and adjustment of scoring weights, is designed to be extensible. As the network evolves more granular performance measurements become available.

5.6. Staking and Reward Distribution

5.6.1. StakeWeight Mechanism

Operators participate in a vote-escrow staking system inspired by veCRV where voting power (StakeWeight) decays linearly over time:

$$StakeWeight = tokens \cdot \frac{lockTime}{maxLockTime}$$

Where:

- *tokens* is the amount of WCT tokens locked
- *lockTime* is the chosen lock duration in weeks
- *maxLockTime* is set to 209 weeks (4 years).

The system begins with a conservative lock duration limit of 104 weeks (2 years). As the network matures, governance can gradually extend this limit up to the maximum of 209 weeks, allowing for longer-term commitment while maintaining consistent stake weight calculations.

Key characteristics:

- Operators lock WCT tokens to generate stake weight
- Stake weight follows a continuous linear decay
- Longer lock periods result in higher initial stake weight
- A minimum of 100,000 stake weight must be maintained at the start of each weekly epoch to qualify for rewards

5.6.2. Dual Reward Structure

Operators earn two types of rewards:

1. **Staking Rewards**: Base rewards for locking WCT tokens, calculated using the same formula as regular stakers: $StakingReward_i = \frac{userStakeWeight_i}{totalStakeWeight} \cdot epochStakingRewards$
2. **Performance Rewards**: Additional rewards based on node performance.

This dual structure incentivizes both long-term commitment through staking and high-quality service through performance.

5.6.3. Reward Calculation

The system employs a straightforward formula that translates performance into rewards:

$$Reward_i = R_{max} \cdot P_i$$

Where:

- $Reward_i$ is the reward for node *i*
- R_{max} is the maximum possible reward for perfect performance
- P_i is the node's performance score (expressed in basis points, where 10,000 represents 1.0)

Each node's reward is calculated independently, creating a non-competitive environment where all operators can achieve maximum rewards through optimal performance.

5.7. Security and Risk Management

The reward system implements several security measures:

1. **Finalization Delay**: 24-hour waiting period after epoch end
2. **Single Claim**: One claim per operator per epoch
3. **Stake Verification**: Continuous minimum stake requirement
4. **Oracle Authorization**: Only authorized oracles can submit metrics

5. **Performance Validation**: Multi-point verification of submitted metrics:
    1. Prevention of future epoch submissions
    2. Enforcement of proper epoch boundaries
    3. Protection against out-of-order submissions
    4. Basic sanity checks on submitted values

5.8. Risk Management

In the initial phase, the system operates without slashing mechanisms to allow operators to familiarize themselves with the network requirements. This conservative approach prioritizes network growth and operator confidence while maintaining high standards through positive incentives rather than penalties.

6. Future Work

6.1. Latency

While the network already demonstrates solid global latency the network is exploring multiple avenues to further reduce latency:

- **Performance-Aware Rendezvous Hashing**: Adjust the node responsibilities based on live performance metrics from the nodes to incentivize better performance.
- **Geo-Aware Routing:** Enhance the hashing logic with geo-performance insights and automatically route requests to topologically closer nodes.
- **Multiple Key Spaces Within the Global WCN Cluster**: Some services (e.g. Smart Sessions) using WCN may be deployed to a single region. For such services, it may be possible to further reduce storage latency by using an alternative key space, where all nodes are located in the same geographical region as the client. The solution here is to have multiple key spaces inside the global WCN cluster. These key spaces may be registered by clients, and have different requirements (e.g. latency-based) and replication strategies (e.g. replication factor and consistency guarantees).

6.2. Availability

In the event of network partitioning, it's possible that some clients may not be able to reach some nodes. To maintain service availability for all clients at all times, the network is exploring the possibility of providing communication tunneling using a separate API of nodes.

If a node becomes unreachable to a client but is otherwise healthy, a client may choose to open a data tunnel through an intermediary node based on the total latency to the target node. Not only would this help improve availability in the event of network partitioning, but it's also possible to achieve lower latency in certain scenarios.

6.3. Incentive Layer

The network's transition toward decentralization requires evolving its economic model in three key areas:

Performance Oracle
The current trusted Oracle system will transition to include cryptographic proofs and cross-validation mechanisms, enabling reliable performance measurement without requiring trust in individual operators. This will move from today's manual reward distribution to automated on-chain distribution with Oracle-based performance tracking.

Access Control
System parameters and controls will gradually transition from foundation management to community governance mechanisms, supporting the network's path to decentralization.

Risk Management
Following thorough community consultation, the system will evaluate introducing graduated penalty mechanisms while maintaining its focus on positive incentives for optimal performance. Additional performance metrics will be introduced to enhance the reward mechanism's ability to incentivize high-quality service.

These evolutions aim to strengthen the self-reinforcing cycle where operators are incentivized to maintain high performance while preserving the system's simplicity and transparency. Each transition will proceed gradually to ensure network stability and security.

6.4. Permissionless

The goal of the system has always been to become fully permissionless. But for anybody to run a node the network will need to most importantly define how the system will detect and penalize malicious behavior. There will be a dedicated technical paper about this published later this year.

Open Enrollment
The network needs to work on the logistics of moving from permissioned enrollment to open enrollment. The network may need to extend node operator staking to allow end-users to delegate stake to node operators as part of this.

Governance and Network Upgrades
The network will need to consider moving parameters such as reward and penalty under DAO control and formalize the logistics of network upgrades.

Trustless Bootstrapping
To become fully permissionless the network may need to iterate on the bootstrapping modes and node identity.

Consensus
Byzantine-Fault tolerance is a critical component of permissionless systems however Raft is not designed to tolerate Byzantine failures.
To address this issue while maintaining the system’s performance guarantees, a new consensus layer is being designed.

Auditing and Accounting
Currently, the network relies on a centralized layer for accounting. Open Enrollment will likely introduce auditing requirements. Both auditing and accounting would need to be on a permissionless ledger themselves for the system to be permissionless.

7. References

[1] Derek Rein, 2022. “WalletConnect v2 Multiplexing Support” https://docs.google.com/document/d/1ArOyT9mkSDk0FL9yCpWogxYPduQ04nzXH0dTaCBoQoM/edit?usp=sharing

[2] Derek Rein, 2023. “Decentralizing the Relay” https://docs.google.com/document/d/1WkJZEjDpEvfQAdJyD8T4w_ttbcBZnsWYph3bbMEvGkc/edit?tab=t.0#heading=h.3z63dy46s8ci

[3] Derek Rein, 2023. “Decentralizing WalletConnect and Why Low-Latency is Hard” https://www.youtube.com/watch?v=wdvfl_Ku-xE

[4] Ongaro, Diego; Ousterhout, John, 2013. “In Search of an Understandable Consensus Algorithm” https://raft.github.io/raft.pdf

[5] Thaler, David; Chinya Ravishankar, 1996. “A Name-Based Mapping Scheme for Rendezvous” https://www.eecs.umich.edu/techreports/cse/96/CSE-TR-316-96.pdf

[6] https://github.com/libp2p/specs/blob/master/peer-ids/peer-ids.md#peer-ids

[7] E. Rescorla, 2018. “The Transport Layer Security (TLS) Protocol Version 1.3” https://tools.ietf.org/html/rfc8446

[8] https://github.com/libp2p/specs/blob/master/tls/tls.md

[9] J. Iyengar, Ed; M. Thomson, Ed, 2021. “QUIC: A UDP-Based Multiplexed and Secure Transport” https://datatracker.ietf.org/doc/rfc9000/