Whitepaper

> **WARNING**
> The SQD whitepaper and any other documents attached or linked to this statement are intended only to provide a broad overview of the general direction of SQD and the SQD Network. The whitepaper and these documents are intended for information purposes only and may not be incorporated into any contract. These materials are not and should not be considered to offer a commitment to deliver any material, code, functionality, token, and should not be relied upon in making token purchase decisions or any kind of investment. All details related to the development, timing, or issue of any SQD feature or token remain at the sole discretion of Subsquid Labs GmbH.

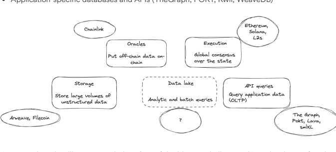
Intro

Blockchain, while initially designed solely for peer-to-peer transfers (Bitcoin), has since evolved into a general purpose global execution runtime. The introduction of smart contract blockchains (Ethereum being the first) has enabled the creation of decentralized Apps (dApps) that are now catering to users in countless verticals, including:

- Decentralized Finance (DeFi) and tokenized real-world assets (RWA)
- Decentralized Social application (DeSoc)
- Decentralized Gaming with peer-to-peer game economies (GameFi)
- Decentralized physical infrastructure (DePIN)
- Decentralized creator economy (Music, Videos, NFTs)

One of the key fundamentals that has allowed countless dApps to thrive is the emergent infrastructure addressing the challenges of scaling, storage, and retrieval of data. We also refer to this as the "Web3 tech stack". The below all form part of the stack:

- Scalable execution (L2 chains, Rollups, Bridges, Solana)
- Bringing off-chain and real-world data on-chain (Oracles like Chainlink)
- Permanent and long-term storage for large data (Arweave, Filecoin)
- Application-specific databases and APIs (TheGraph, POKT, Kwil, WeaveDB)



However, there is still one part missing. One of the biggest challenges dApp developers face is accessing data at scale. It's currently highly complex to query and aggregate the exponentially growing amount of data produced by blockchains (transactions and state data), applications (decoded smart contract states), and any relevant off-chain data (prices, data kept in IPFS and Arweave, pruned EIP4844 data). In Web2, data produced by applications (such as logs or client data) is stored in centralized data lakes like BigQuery, Snowflake, Apache, and Iceberg to facilitate access. Storing Web3 data in similar centralized data lakes would defeat the purpose of open, resilient access. Still, Web3 app data, if aggregated, filtered, and easily extractable, will serve as a catalyst to unlock the potential of the industry in a multi-chain paradigm.

Instead of replicating Web2 approaches the permissionless and decentralized nature of Web3 requires a fundamentally different solution with the following properties:

- Infinite horizontal scalability: the capacity of the lake should grow indefinitely as new nodes join
- Permissionless data access: the data can be uploaded and queried without any gatekeeping or centralized control
- Credible neutrality: no barrier for new nodes to join the network
- Trust-minimized queries: the data can be audited, and all clients can verify the query result
- Low maintenance cost: the price per query should be negligible

The SQD Network is built to satisfy all the above properties, thanks to its architecture where:

- The raw data is uploaded into permanent storage by the data providers, which act as oracles but for "big data"
- The data is compressed and distributed among the network nodes
- Node operators have to bond a security deposit, which can be slashed for byzantine behavior
- Each node efficiently queries the local data with DuckDB
- Any query can be verified by submitting a signed response to an on-chain smart contract

With the recent advances in in-process databases like DuckDB and an ever-decreasing price of quality SSD storage, the operational cost of a single node handling 1Tb of data is as low as 35/mo [1] and will likely decline further in the future.

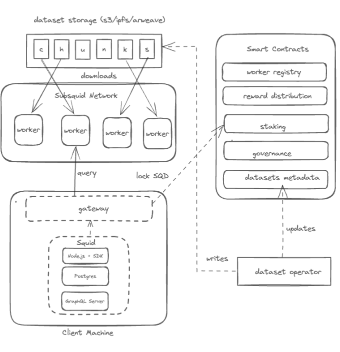
In the first stage, the SQD Network only supports simple queries that can be resolved by querying the local data of a suitable node. This already provides a fundamentally new primitive for on-chain data availability, fully eliminating the need to maintain expensive archival blockchain nodes to access historical blockchain data.

After the bootstrapping period, the network will be expanded with an additional query planning and execution layer supporting general-purpose SQL queries inspired by Apache Calcite and DataFusion.

Design Overview

The below actors all participate in the SQD Network:

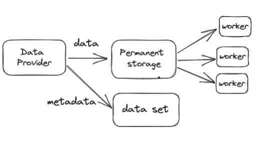
- Data Providers
- Workers
- Scheduler
- Logs Collector
- Rewards manager
- Data Consumers



Data Providers

Data Providers produce data to be served by SQD Network. Currently, the focus is on on-chain data, and data providers are blockchains and L2s. At the moment, SQD only supports EVM- and Substrate chains, but we plan on adding support for Cosmos and Solana in the near future.

Data providers are responsible for ensuring the quality and timely provision of data. During the bootstrapping phase, Subsquid Labs GmbH acts as the sole data provider for the SQD Network, serving as a proxy for chains from which the data is ingested block-by-block. The ingested data is validated by comparing hashes. It's then split into small compressed chunks and saved into persistent storage, from which the chunks are randomly distributed between the workers.



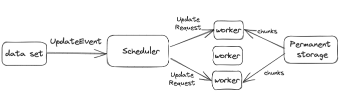
The metadata is saved on-chain and is updated by the data provider each time a new piece of the data is uploaded. The metadata describes the schema, the data layout, the reserved storage, and the desired replication factor to be accommodated by the network. You can find the metadata structure in the Appendix.

The data provider is incentivized to provide consistent and valid data, and it is their responsibility to make the data available in persistent storage. During the bootstrap phase, the persistent storage used by Subsquid Labs is an S3-compatible service with backups pinned to IPFS. As the network matures, more data providers and storage options will be added, with data providers being vetted by on-chain governance and known trusted parties.

Data providers pay on-chain subscription fees to the network to make data available and to have it served by workers. These fees are sent to the SQD Network treasury.

Scheduler

The scheduler is responsible for distributing the data chunks submitted by the data providers among the workers. The scheduler listens to updates of the data sets, as well as updates of the worker sets, and sends requests to the workers to download new chunks and/or redistribute the existing data chunks based on the capacity and the target redundancy for each dataset. Once a worker receives an update request , it downloads the missing data chunks from the corresponding persistent storage.



Workers

Workers contribute the storage and compute resources to the network. They serve the data in a peer-to-peer manner for consumption and receive $SQD tokens as compensation. Each worker has to be registered on-chain by bonding `100_000` SQD tokens, which can be slashed if the worker provably violates the protocol rule. $SQD holders can also delegate to a specific worker to signal the reliability of the worker and earn a portion of the rewards.

The rewards are distributed each epoch and depend on:

- The previous number of epochs the worker stayed online
- The amount of data served to clients
- The number of delegated tokens
- Fairness
- Liveness during the epoch

Workers submit ping information along with their signed logs of the executed query requests to the Validator, thus committing to the results returned to the clients.

Logs collector

The Logs Collector's sole responsibility is to collect the liveness pings and the query execution logs from the workers, batch them, and save them into public persistent storage. The logs are signed by the workers' P2P identities and pinned to IPFS. The data is stored for at least 6 months and used by other network participants.

Reward Manager

The Reward Manager is responsible for calculating and submitting worker rewards on-chain for each epoch. The rewards depend on:

- Worker liveness during the epoch
- Delegated tokens
- Served queries (in bytes; both scanned and returned sizes are accounted for)
- Liveness since the registration

The Reward Manager accesses the logs, calculates the rewards, and submits a claimable commitment on-chain for each epoch. Each worker then claims their reward individually. The rewards may expire after an extended period of time

Data Consumers

To query the network, data consumers have to operate a gateway or use an externally provided service (public or private). Each gateway is bound to an on-chain address. The number of requests a gateway can submit to the network is capped by a number calculated based on the amount of locked $SQD tokens. Consequently, the more tokens are locked by the gateway operator, the more bandwidth it is allowed to consume.

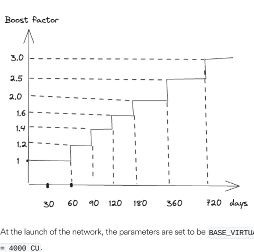
One can think of this mechanism as if the locked SQD yields virtual "compute units" (CU) based on the period the tokens are locked. All queries cost the same price of 1 CU (until complex SQL queries are implemented).

The query cap is calculated by:

- Calculating the virtual yield in SQD on the locked tokens (in SQD)
- Multiplying by the current CU price (in CU/SQD)

Boosters

The locking mechanism has an additional booster design to incentivize gateway operators to lock their tokens for longer periods of time for an increase in CU. The longer the lock period, the more CUs are allocated per SQD/yr. The APY is calculated as `BASE_VIRTUAL_APY * BOOSTER`.



At the launch of the network, the parameters are set to be `BASE_VIRTUAL_APY = 12%` and `1SQD = 4000 CU`.

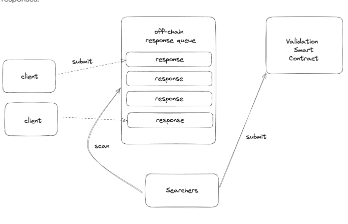
For example, if a gateway operator locks 100 SQD for 2 months, the virtual yield is 2SQD, which means it can perform 8000 queries (8000 CU). If 100 SQD are locked for 3 years, the virtual yield is `12% * 3` APY, so the operator gets CUs worth 108 of SQD, that is it can submit up to `432000` queries to the network within the period of 3 years.

Query Validation

The SQD Network provides economic guarantees for the validity of the queried data, with the added possibility of validating specific queries on-chain. All query responses are signed by the worker who executed the query, acting as a commitment to the query response. Anyone can submit such a response on-chain, and if it is deemed incorrect, the worker bond is slashed. The smart contract validation logic may be dataset-specific depending on the nature of the data being queried, with the following options:

- Proof by Authority: a white-listed set of on-chain identities decides on the validity of the response.
- Optimistic on-chain: after the validation request is submitted, anyone can submit a claim proving the query response is incorrect. For example, assuming the original query was "Return transactions matching the filter `X` in the block range `[Y, Z]`" and the response is some set of transactions `T`. During the validation window, anyone can submit a Merkle proof for some transaction `t` matching the filter `X` yet not in `T`. If no such proofs are submitted during the decision window, the response is considered valid.
- Zero-Knowledge: a zero-knowledge proof that the response exactly matches the requests. The zero-knowledge proof is generated off-chain by a prover and is validated on-chain by the smart contract.

Since submitting each query for on-chain validation on-chain is costly and not feasible, clients opt-in to submit query responses in an off-chain queue, together with the metadata such as response latency. Then, independent searchers scan the queue and submit suspicious queries for on-chain validation. If the query is deemed invalid, the submitter gets a portion of the slashed bond as a reward, thus incentivizing searchers to efficiently scan the queue for malicious responses.



SQD Token

SQD is the ERC-20 protocol token that is native to the SQD Network ecosystem. The token smart contract is to be deployed on the Ethereum mainnet and bridged to Arbitrum One. This strategy seeks to ensure the blockchain serves as a reliable, censorship-resistant, and verifiably impartial ledger, facilitating reward settlements and managing access to network resources.

The SQD token is a critical component of the SQD ecosystem. Use cases for the SQD token are focused on streamlining and securing network operations in a permissionless manner:

1. Alignment of incentives for infrastructure providers: SQD is used to reward node operators that contribute computation and storage resources to the network.
2. Curation of network participants: Via delegation, the SQD token design includes built-in curation of nodes, facilitating permissionless selection of trustworthy operators for rewards.
3. Fair resource consumption: By locking SQD tokens, consumers of data from the decentralized data lake may increase rate limits.
4. Network decision making: SQD tokenholders can participate in governance, and are enabled to vote on protocol changes and other proposals.

The SQD token's innovative curation component allows the SQD community to delegate SQD to Node Operators of their choice, ensuring trustlessness. SQD's utility as a tool for adjusting rate limits is unique in increasing trustless performance, by locking SQD tokens, without having to pay a centralized provider for quicker or more efficient data access.

Appendix I -- Metadata

The metadata has the following structure:

```
interface DataSet {
  dataset_id: string
  // how many times the dataset should be replicated
  replication_factor: int
  // a unique URL for permanent storage. Can be s3:// bucket
  permanent_storage: string
  // how much space should for a single replica.
  // the total space required by a dataset is replication_factor * reserved
  reserved_space: int
  // dataset-specific additional metadata
  extra_meta?: string
  // dataset-specific state description
  state?: string
  // reserved for future use, must be false
  premium: boolean
}
```

The on-chain state of the dataset should be updated by the data provider periodically, and the total space must not exceed `reserved_space`. The dataset can be in the following states:

```
enum DatasetState {
  // submitted by the data provider
  SUBMITTED,
  // the data being distributed by the network
  IN_PREPARATION,
  // served by the network
  ACTIVE,
  // if the subscription fee
  // has not been provided for the duration of an epoch
  COOLDOWN,
  // if the dataset is no longer served by the network
  // and will be deleted
  DISABLED
}
```

The `Scheduler` changes the state to `IN_PREPARATION` and `ACTIVE` from `SUBMITTED`. The `COOLDOWN` and `DISABLED` states are activated automatically if subscription payments aren't made.

At the initial stage of the network, the switch to disabling datasets is granted to Subsquid Labs GmbH, which is going to be replaced by auto payments at a later stage.

Appendix II -- Rewards

The network rewards are paid out to workers and delegators for each epoch. The Reward Manager submits an on-chain claim commitment, from which each participant can claim.

The rewards are allocated from the rewards pool. Each epoch, the rewards pool unlocks `APY_D * S * EPOCH_LENGTH` in rewards, where `EPOCH_LENGTH` is the length of the epoch in days, `S` is the total (bonded + delegated) amount of staked `SQD` during the epoch and `APY_D` is the (variable) base reward rate calculated for the epoch.

Rewards pool

The `SQD` supply is fixed for the initial pool, and the rewards are distributed from a pool, to which `10%` of the supply is allocated at TGE. The reward pool has a protective mechanism which caps the amount of rewards distributed per epoch, and it is subject to change via governance.

During the initial 3-year bootstrapping phase, the reward cap and the total supply of `SQD` is fixed. Afterwards, the reward cap drops significantly until the governance motion concludes on the inflation schedule and new tokens are minted to replenish the reward pool.

Unlike most projects who fix the inflation schedule before the launch, postponing this decision leaves a lot more flexibility and allows the community to analyze the historical 3-year data to make and informed decision on the future issuance of `SQD`.

Reward rate

The reward rate depends on two factors: utilization of the network and staked supply. The network utilization rate is defined as

```
u_rate = (target_capacity - actual_capacity)/target_capacity
```

The target capacity is calculated as

```
target_capacity = sum([d.reserved_space * d.replication_factor])
```

where the sum is over the non-disabled datasets `d`.

The actual capacity is calculated as

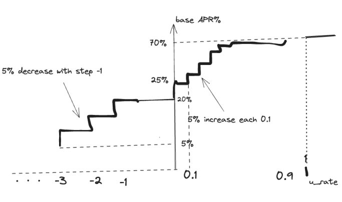
```
actual_capacity = num_of_active_workers() * WORKER_CAPCITY * CHURN
```

The `WORKER_CAPACITY` is a fixed storage per worker, set to `1TB`. `CHURN` is a discounting factor to account for the churn, set to `0.9`.

The target APR (365-day-based rate) is then calculated as:

```
rAPR = MIN(base_apr(u_rate), APR_CAP(total_staked))
```

The `base_apr` is projected to be around `20%` in the equilibrium state, when the actual worker capacity matches the desired network capacity, set externally. It is increased up to `70%` to incentivize more workers to join the network until the target capcity is reached:



The `APR_CAP` cut-off is added to cap the total rewards per epoch. One defines first

```
total_staked = bonded + delegated
```

to be the total amount of reward-earning `SQD` locked by workers (`bonded`) and delegators (`delegated`) respescitvely.

We then define

```
APR_CAP(total_staked) = 0.3 * INITIAL_POOL_SIZE / total_staked
```

It is set so that after the 3 year boostrapping period at most 90% of the initial reward pool is spent.

Worker reward rate

For each epoch, `rAPR` is calculated, and the total of

```
R = rAPR/365 * total_staked * EPOCH_LENGTH
```

is unlocked from the rewards pool.

The rewards are then distributed between the workers and delegators, and the leftovers are split between the burn and the treasury.

For a single worker and stakers for this token, the maximal reward is `rAPR/365 * (bond + staked) * EPOCH_LENGTH`. It is split into the worker liveness reward and the worker traffic reward.

Let `S[i]` be the stake for `i`-th worker and `T[i]` be the traffic units (defined below) processed by the worker. We define the relative weights as

```
s[i] = S[i]/sum(S[i])
t_scanned[i] = T_scanned[i]/sum(T_scanned[i])
t_e[i] = T_e[i]/sum(T_e[i])
t[i] = sqrt(t_scanned[i] * t_e[i])
```

In other words, `s[i]` and `t[i]` correspond to the contribution of the `i`-th worker to the total stake and to total traffic, respectively.

The traffic weight `t[i]` is a geometric average of the normalized scanned (`t_scanned[i]`) and the egress (`t_e[i]`) traffic processed by the worker. It is calculated by aggregating the logs of the queries processed by the worker during the epoch, and for each processed query, the worker reports the response size (egress) and the number of scanned data chunks.

The max potential yield for the epoch is given by `rAPR` described above:

```
r_max = rAPR/365 * EPOCH_LENGTH
```

The actuall yield `r[i]` for the `i`-th worker is discounted:

```
r[i] = r_max * D_liveness * D_traffic(t_i, s_i) * D_tenure
```

`D_traffic` is a Cobb-Douglas type discount factor defined as

```
D_traffic(t_i, s_i) = min( (t_i/s_i)^alpha, 1 )
```

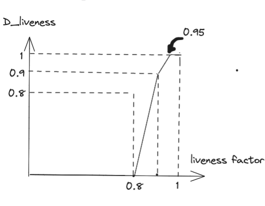
with the elasticity parameter alpha set to `0.1`.

It has the following properties:

- Always in the interval `[0, 1]`
- Goes to zero as `t_i` goes to zero
- Neutral (i.e., close to 1) when `s_i ~ t_i`, that is, the stake contribution is fair (proportional to the traffic contribution)
- Excess traffic contributes only sub-linearly to the reward

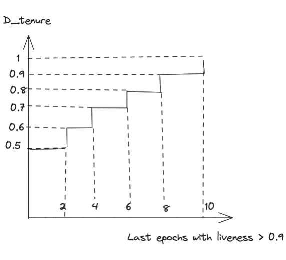
`D_liveness` is a liveness factor calculated as the percentage of the time the worker is self-reported as online. A worker sends a ping message every 10 seconds, and if there are no pings within a minute, the worker is deemed offline for this period of time. The liveness factor is the percentage of the time (with minute-based granularity) the network is live. We suggest a piecewise linear function with the following properties:

- It is `0` below a reasonably high threshold (set to `0.8`)
- Sharply increases to near `1` in the "intermediary" regime `0.8-0.9`
- The penalty around `1` is diminishing



Finally, `D_tenure` is a long-range liveness factor incentivizing consistent liveness across the epochs. The rationale is that

- The probability of a worker failure decreases with the time the worker is live thus freshly spawned workers are rewarded less
- The discount for freshly spawned workers discourages the churn among workers and incentivizes longer-term commitments



Distribution between the worker and delegators

The total claimable reward for the `i`-th worker and the stakers is calculated simply as `r[i] * s[i]`. Clearly, `s[i]` is the sum of the (fixed) bond `b[i]` and the (variable) delegated stake `d[i]`. Thus, the delegator rewards account for `r[i] * d[i]`. This extra reward part is split between the worker and the delegators:

- The worker gets: `r[i] * b[i] + 0.5 * r[i] * s[i]`
- The delegators get `0.5 * r[i] * s[i]`, effectively attining `0.5 * r[i]` as the effectual reward rate.

The rationale for this split is:

- Make the worker accountable for `r[i]`
- Incentivize the worker to attract stakers (the additional reward part)
- Incentivize the stakers to stake for a worker with high liveness (and, in general, high `r[i]`)

At an equilibrium, the stakers will get a `10%` annual yield, while workers get anything between `20-30%` depending on the staked funds. Note that the maximal stake is limited by the bond size.

References

[1] SuperDAO Growth Trends report

[2] Based on the estimate that read-only RPC queries constitute roughly 90% of the RPC provider traffic