Sizzler Network

A self-sustaining network of keepers on Tezos

Author: Anshu Jalan (anshujalan.xyz)

Smart contracts are inherently event-driven and can neither communicate directly with the outside world nor execute automated tasks, thus requiring developers to handle the events by setting up custom centralised infrastructure. Sizzler Network is a decentralised system built on top of Tezos blockchain that enables the delegation of such tasks to bonded entities called 'Sizzlers'. It aims to function as a self-sustaining system through effective incentivisation and reward mechanics using its utility token called Sizzle.

1. Introduction

Complex smart contract systems often require actions to be taken automatically when the storage reaches a certain state. For instance, a smart contract system representing decentralised derivatives would ideally need automated margin calls and closing of positions. However, this cannot be implicitly done on-chain due to the way smart contracts work on Tezos (or any other existing blockchain).

As a resolution, developers have to set up extensive dev-ops infrastructure to handle such tasks through external smart contract calls. This is often tedious and leads to launch delays. As a side-effect, it also tends to create a centralised point of failure, leading to untimely events and poor user experience if and when the infrastructure goes down.

Sizzler Network aims to solve this problem by providing a system to which dev-ops tasks can be off-loaded and left to be handled by entities called 'Sizzlers'. These Sizzlers are rewarded and optionally incentivised through tips, using the network's utility token called 'Sizzle' (SZL).

2. Tasks

A 'task' refers to an action that is to be taken by a Sizzler, usually on a recurring basis. The action involves calling the entrypoint of a smart contract deployed on

Tezos. This may either be a simple parameter-less entrypoint requiring a standard call at regular intervals (eg: epoch updates) or it may require parameters that are to be calculated by setting up custom off-chain logic (eg: finding IDs of positions that can be liquidated on a derivatives platform).

Tasks are to be set up by the developers of a decentralized application and sent over to Sizzler Network's governance system (explained later) for approval.

Programmatically, a task is represented by 5 fields:

- Tezos address of the task contract.
- Entrypoint to be called in the task contract.
- An IPFS string that points to the description of the task.
- Tezos address of the task owner.
- The tip amount (in SZL), given to the sizzler for task completion.

The entrypoint in the task contract has to be written in a certain way for it to be approved by Sizzler Network. It must involve calling the **complete_task** entrypoint in the **TaskManager** contract. This call can either be built into the core contracts of a platform, or a dedicated proxy contract can be set up (recommended). Examples for both methods are given below in Smartpy

a. **In-built call**

```
def liquidate_position(self, position_id):
  sp.set_type(position_id, sp.TNat)

  # Logic to liquidate the associated position
  # ....

  # Required call to the TaskManager contract
  c = sp.contract(
    sp.TAddress,
    self.data.task_manager_address,
    "complete_task"
  ).open_some()
  sp.transfer(sp.sender, sp.tez(0), c)
```

b. Proxied call (recommended)

```
def liquidate_position_proxy(self, position_id):
  sp.set_type(position_id, sp.TNat)

  # Call to core contract
  c_core = sp.contract(
    sp.TNat,
    self.data.core_address,
    "liquidate_position",
  ).open_some()
  sp.transfer(position_id, sp.tez(0), c)

  # Required call to the TaskManager contract
  c_task_manager = sp.contract(
    sp.TAddress,
    self.data.task_manager_address,
    "complete_task"
  ).open_some()
  sp.transfer(sp.sender, sp.tez(0), c_task_manager)
```

It is to be noted that the entrypoint that includes this internal call can only be invoked by a Sizzler.

3. Sizzlers

Sizzlers form the core of the network and are responsible for completing the tasks delegated to the system. Besides, they are also responsible for handling the governance i.e submitting proposals and voting on them.

a. Bonding

To become a Sizzler, one has to go about the ‘skin in the game’ route. The participant would provide liquidity to a Quipuswap SZL-kUSD pool, and stake the

subsequently received LP tokens (abbreviated as tQPLP) on Sizzler Network (specifically in the **SizzlerManager** contract). There is a minimum bonding amount that is stored as a parameter in the storage.

b. Task Limits

To ensure a level playing field for all, there is a limit on the number of tasks each sizzler can perform in a given time interval (storage parameter).

The limit is calculated as $amount\ of\ tQPLP\ staked\ /\ minimum\ bond\ value$. Therefore, if the minimum bond value is let's say 1.5 tQPLP tokens, then a Sizzler who wants to perform 5 tasks in each interval must stake at least 9 tQPLP tokens.

There is a task counter within the contract keeping track of the number of tasks you have completed in the current interval. It resets to 0 once the interval is over.

c. Deposit and Withdrawal Timeline

The amount of staked tQPLP can be changed by the Sizzler. Once a deposit or a withdrawal of LP tokens is initiated, the Sizzler needs to wait for a ‘delay period’ (storage parameter, separate for deposit and withdrawal). Only after this period is over, will the Sizzler have a higher staked value (for deposit), or receive their tokens back to their wallet (for withdrawal).

4. Crypto-Economics

Since the network has its own utility token ‘Sizzle’ (SZL), there are various crypto-economic sides associated with it. SZL is based on the FA1.2 standard on Tezos.

a. Minting

SZL can only be minted by a dedicated **Minter** contract. SZL comes into existence through two methods- timed emissions and minting proposals.

- **Timed Emissions:** The Minter contract has a provision for two time-based emissions, represented as $X\ Sizzle\ to\ be\ minted\ every\ T\ seconds$. These values can be changed through the governance system.
    - **Sizzler Reward Emission:** Whenever a Sizzler completes a task, they are rewarded with accumulated SZL being generated through this emission channel. This is designed such that the Sizzlers have an incentive to do tasks in a competitive way, even in periods of low activity.
      The reward is calculated simply as
      $Emission\ rate\ \ *\ \ Time\ passed\ since\ last\ task\ completed\ on\ the\ system$
    - **Dev Share:** An optional dev share emission may be set to generate working capital for the dev team.
- **Minting Proposals:** A proposal to explicitly mint SZL for treasury, grants etc, can be submitted to the governance system, and be approved by the Sizzlers.

b. Tipping

Task owners can tip the Sizzlers in SZL for every task completed. Tipping is useful if the owners want the Sizzlers to prioritise their tasks in times of high activity. Additionally, tips can help with incentivising the Sizzlers when the task is complex and requires extensive off-chain work.

Tipping is essentially a two-step process. Owners can set the tip value in SZL for each completion, and then add ‘credits’ to the task. The tip amount is deducted from the credits when a task is completed.

c. **Liquidity**

Liquidity is highly important for a system like Sizzler Network, where during times of high activity SZL must be liquid enough to keep up with the demand for tipping.

This is the motivating factor behind using SZL-kUSD dex pair LP token as the bonding medium and making it have a proportional relationship with the task limit i.e *more liquidity provided equals more tasks available.*

5. Governance

The governance of Sizzler Network is handled by the **Governor** contract. It functions like a DAO, with the difference that instead of having a token as the carrier of voting power, it instils the power in the sizzlers with the voting weight being equal to their current task limit. This adds to the value of liquidity since *more liquidity provided equals more tasks available, which further equates to higher voting power.*

The Governor has the standard governance parameters like quorum threshold, proposal threshold, voting and timelock period. The proposal threshold value represents the minimum task limit that a Sizzler requires in order to submit a new proposal.

A simple majority of yes votes over no votes is used to settle proposals that have attained quorum.

The major use of the governance system would be the addition of tasks to the network. Task owners would need to submit a new task request through a sizzler and have it approved by others. This prevents spam and malicious tasks from clogging the system.

6. Use-cases

Sizzler Network is designed for tasks that are either inherently permissionless i.e not requiring admin access or tasks for which the owner is trusting the decentralized network of Sizzlers to maintain integrity. Accordingly, potential use-cases would be

- Automated margin calls and closing of positions on a decentralised derivatives platform.
- Simple recurring tasks like epoch updates.
- As an external source of randomness. Preferably using verifiable functions.
- As an oracle. However, the users must implement fallback and dispute mechanisms for safety.