**Transaction Broadcaster Service**

By: Poh Jing Choon

According to my understanding of a transaction broadcaster service, it is a service that manages the *receiving* and *sending* of transactions across other entities. Upon receiving a transaction, the service will sign it first, before broadcasting it to an EVM-compatible blockchain network. To fulfil the requirements stated, the overlying structure will consist of the following components:

1. **Database** to store current transaction and past transactions managed.
2. **Processor** – to sign the transactions.
3. **Dispatcher** – to broadcast signed transactions out to the network.

The entities in the network are both acting as the client and server in this network. The process begins when an entity sends a transaction to our current entity through the exposed internal API.

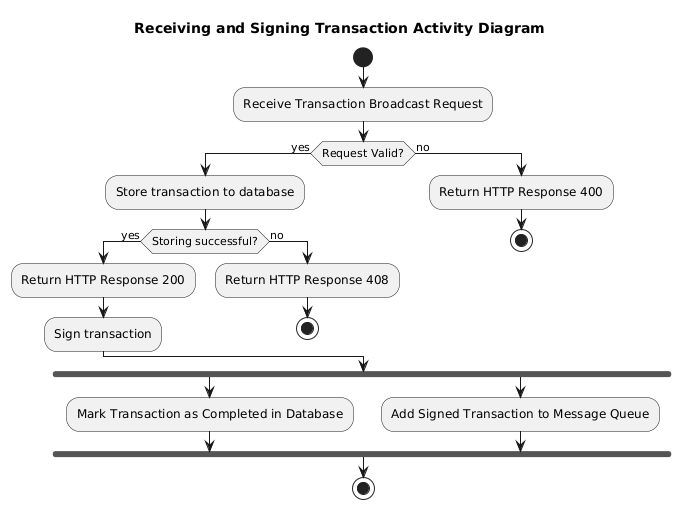
A diagram of a service

AI-generated content may be incorrect.

High Level Architecture Diagram of the main entities (with Queue)

Receiving and signing the transaction

Following are the actions taken up by the processor/client upon receiving the transaction.

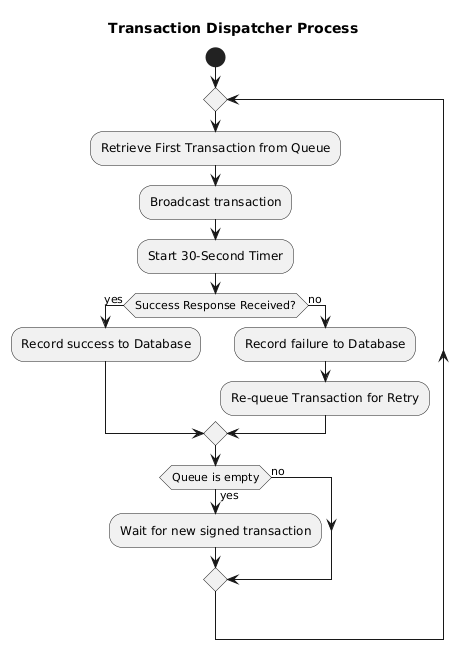


* It is noteworthy for the client to ensure the transaction is stored in the database first, so that in the event the broadcaster service restarts unexpectedly, the processor will still be able to access the transaction after reboot.

After completing the signing, the signed transaction will be enqueued to a signed-transaction queue, and it is managed by the **Dispatcher** from here.

Sending the signed transaction

The dispatcher is responsible to ensure the signed transaction is sent out successfully. It retrieves the first transaction in the queue, and makes an RPC request it to the EVM-compatible blockchain network. After which, it starts a timer up to 30 seconds. If it receives a success code before the timer expires, the dispatcher will record this success broadcast in the database and move on to broadcast the next signed transaction in the queue. Else, if it receives a failure code or timer expired, it will note this down in the database and broadcast the signed transaction again.



Activity diagram for illustration.

Entities and their rough proposed supported methods

**Database** – CRUD operations

* Possible fields in the DB may include: Transaction ID, data, receiver address, etc.

**Processor**

* .signTransaction(Transaction)
* .validateAndParse(Transaction)
* Returning of HTTP Responses
* CRUDing to database
* .enqueue(SignedTransaction)

**Dispatcher**

* .dequeue()
* .broadcast(SignedTransaction, network)
* .startTimer() / .resetTimer() maybe
* CRUDing to database
* .broadcastImmediate(SignedTransaction) [ADMIN PRIVILEGED]

Possible Extensions

1. The dispatching of signed transactions can be done all at once, with instantiation of multiple timers and parallel flows.
2. Setting up of different account types to better facilitate admin and normal users activities.