Problem 6: Transaction Broadcaster Service

Understanding the Problem

In this problem, we are interested in developing a transaction broadcaster service that broadcast blockchain transactions to other nodes.

In developing such a system, it is essential to determine the goals. In this case, I believe that the goals are:

1. If a correct (non-faulty) node sends a signed transaction message, then the EVM-compatible blockchain network will agree on the delivery of the signed transaction message.
2. There is a guarantee that transactions are consistently delivered to all nodes.
3. If a transaction fails, the transaction can be retried.

To achieve such a system, the components we need are:

- Reliable delivery system: To ensure that the message is delivered to every node, **when broadcasting fail**
- Best effort broadcast system: A node delivers message to other nodes. A correct node would be delivered the message.
- Failure detector: To **detect transactions that have failed** and **retry transactions when failed**

System Design

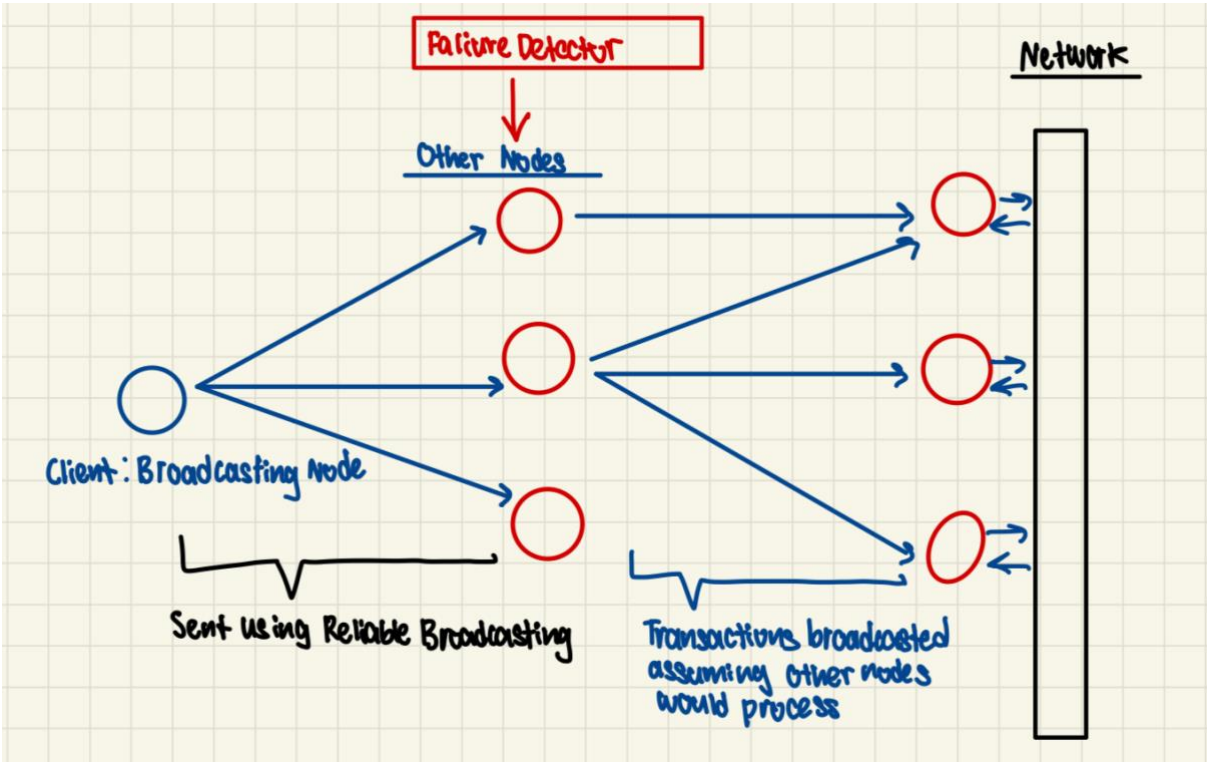

Figure 1: Transaction Broadcaster Service Overview

Steps:

1. The broadcasting service sends the transaction to other nodes on the network. Here we assume that, every correct node would be eventually delivered the message.

2. The failure algorithm detects nodes that might have failed. It is given that we can make a RPC request to the node, to determine whether the node has failed.

   The failure algorithm could be designed to query the blockchain node every 20s to determine whether a process has failed. The rationale for choosing 20s is that it is given that 95% of the time it would return with a success code within 20s. Hence, we can assume that the response above 20s is likely to mean that the process has failed.

3. Once the failure algorithm detects a failure, it re-broadcasts the message. The entire cycle continues.

With this lazy reliable broadcast service, we can ensure that every correct transaction gets broadcasted. Upon failure, we can re-broadcast the transaction.

As a result, this would require that the failure algorithm keeps track of all delivered messages and undelivered messages. As the number of transactions increase, it might be resource intensive to keep track of **all messages.**

To reduce the storage, we could implement a function in the failure detector to query the block explorer. Once a transaction has been accepted by the nodes, it would be added to the

chain. By querying the block explorer and matching the transaction hash we can remove processes from the failure detector because those messages have already been transmitted.

We can create an API to query the messages in the failure detector to determine the list of transactions that have failed. This is because we can assume that by removing messages attached to the block, we have a list of messages that have yet to be broadcasted.

Conclusion

In conclusion, we can achieve a transaction broadcaster service using a lazy reliable broadcaster service. This is because it ensures that:

- Every **correct transaction** is delivered to all nodes
- The system is **consistent**. By having a failure algorithm that consistently detects for failure, ensures that the system is predictable.
- The system **scales efficiently**. This is done by having efficient ways of managing storage.