CS765 Spring 2023 Semester
Project Part-3
Building a Layer-2 DAPP on top of Blockchain

Harsh Shah (200050049)
Sahil Mahajan (200050124)
Shubham Bakare (200050133)

Contents

1 Introduction

In this assignment, we have implemented a Level-2 decentralized application (Dapp) on top of Blockchain. It runs on a peer-to-peer decentralized network, providing a platform for users to collectively own and manage the Dapp's smart contract. The Dapp has been deployed on an Ethereum node using Truffle/Ganache and tested to ensure its functionality works as intended. Users can access the Dapp and its features together through joint accounts, allowing for a collaborative and decentralized approach to managing the Dapp.

2 Connected Network

To generate a connected network between the users we have used Barabasi Albert graph in the networkx library in Python. The model starts with a small connected graph and iteratively adds nodes to the network. When a new node is added, it connects to m existing nodes with a probability proportional to their degree. This means that nodes with a higher degree are more likely to be selected as connection targets for new nodes, resulting in the creation of hubs in the network.
Below is a sample graph generated using Barabasi-Albert model for 100 nodes:

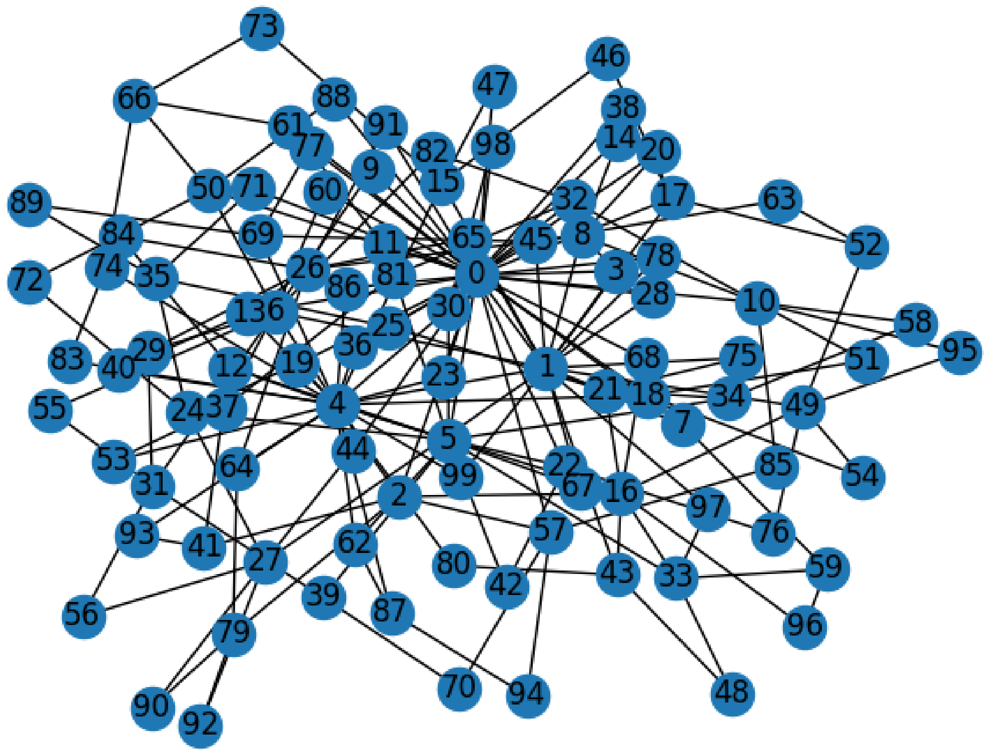

Figure 1: Network Graph

3 Code Implementation

We have implemented the code in two files *Payment.sol* which is the solidity file containing all function implementations and *client.py* containing graph generation, pathfinding, and simulation through the Web3 package of Python on the Truffle/Ganache client.

- **Payment.sol:** It contains the 4 functions - registerUser, createAcc, sendAmount, and closeAccount implemented as per the definitions in the problem statement using various solidity functions and data structures. In the sendAmount function, we send 1 coin from one neighbor to another
- **client.py:** We first connect to the local Ethereum blockchain and connect the contract JSON file with Web3 packages. Then create a connected graph network between nodes using the networkx package in the way described in Section 2. We call all the functions created in the solidity file in the order specified:
  - Register 100 nodes by calling the registerUser function
  - Form a joint account between them such that the combined balance in the account for each pair of nodes follows the exponential distribution with a mean of 10 and is distributed to two users involved in the account equally
  - To run a transaction we select two nodes at random and compute the shortest path between them in the network using BFS algorithm and then call the sendAmount function for each pair of neighbors along this path. In case the function fails at any point in between we note this as a failed transaction, reverse the transaction from this failed point till the start and move on to the next transaction
  - Fire 1000 such random transactions and find the ratio of failed transactions to the total number of transactions
  - On call of closeAccount function, we check if edge is present between the two nodes given. If present check if the graph is still connected after removal of this edge remove it else revert back to previous network graph.

4 Observations

Ratio of Successful Transactions

We have run 1000 transactions and for every 100 transactions, we found the ratio of successful transactions to total transactions.
Ratio values (in percentage) obtained are: [49, 37, 38, 39, 38, 31, 29, 40, 36, 39]
The plot thus obtained of this ratio is below.

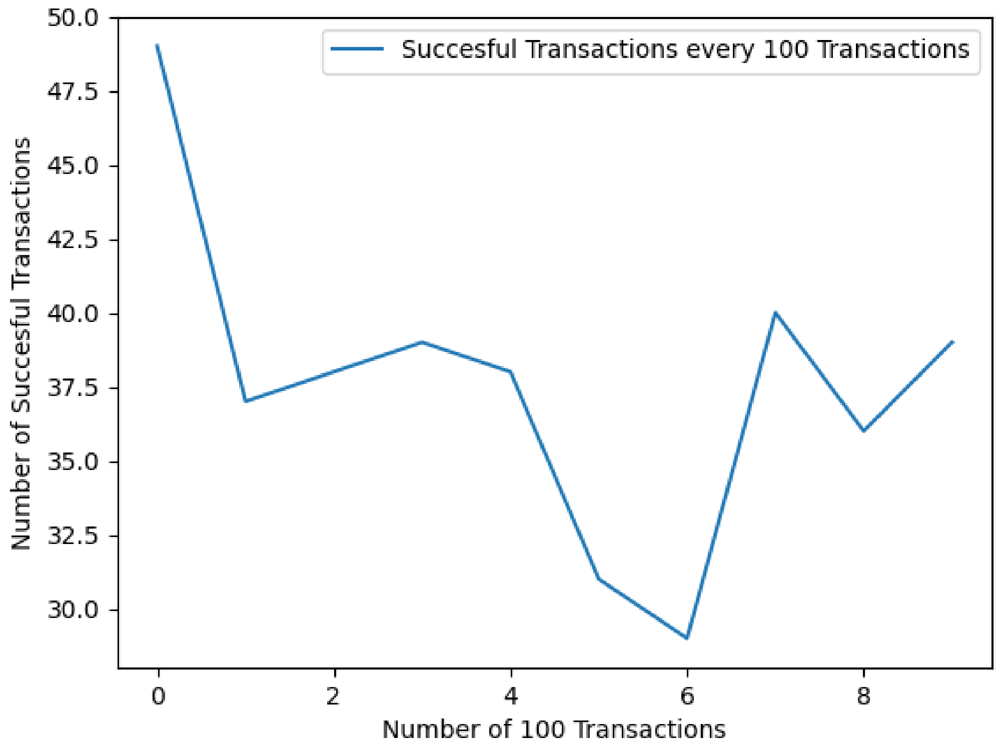

Figure 2: Successful transaction ratio plot

We observed a general trend of increase in number of failed transactions as the total number of transactions increased. We speculate the reason behind this due to power law distribution graph topology. Because few nodes have high degree, there is a high chance that they are included in shortest path between two nodes. Due to this load on few nodes, there would exists some period during which the high degree node is devoid of money on one of the edges. This results in failure of transactions, and increase in such failures as number of transactions increases, since probability of such event happening increases.