# CS 765: Simulation of a P2P Cryptocurrency Network

Advait, Ashwin, Harshvardhan
200100014, 200050016, 200050050

April 2023

## 1 Code Overview

As a part of this assignment, we have implemented the four functions which were mentioned in the problem statement in Solidity programming language. Before describing each function, we would like to first give an idea about the data structures that we have used for this assignment

### 1.1 Data Structures

#### 1.1.1 Node

Node is a struct which contains two fields :-

1. User ID - Refers to the unique id associated with each user
2. User name - Refers to the name of the user

Node struct is used to represent a user in the network. We maintain a global list of all the nodes present in the network.

#### 1.1.2 Edge

Edges is an array which consists information on payment channels between nodes. It is made up of dynamic arrays for each node containing a list of nodes connected to it. Edges are used to represent a joint account between two users. It can be thought of as a directed edge from one user to the other user. The bal array is a 2D array that refers to the balance of that user who is present at the tail of the directed edge. Therefore, when you create a joint account between two users A and B, we create two edges and update the bal array :-

1. User A $\rightarrow$ User B with balance associated with user A
2. User B $\rightarrow$ User A with balance assoicated with user B

### 1.2 Functions

#### 1.2.1 registerUser

This function takes two parameters (user id, user name). When this function is called, we create a new node and initialize it with the given parameters. We add this new node to the global list of nodes.

#### 1.2.2 createAcc

This function takes three parameters (user id 1, user id 2, balance). When this function is called, we create two edges as mentioned in the Edge subsection. We then add these edges to the global list of edges. Here, we have assumed that the balance is equally spilt between the two users.

#### 1.2.3 closeAccount

This function takes two parameters (user id 1, user id 2). When this function is called, we set the balances of both the edges associated with these two nodes to 0 effectively rendering the edge between them as useless.

#### 1.2.4 sendAmount

This function takes two parameters (user id 1, user id 2). When this function is called, we send 1 coin from user 1 to user 2 via the shortest path between them present in the network. We compute the shortest path between the two nodes using Breadth First Search Algorithm. Here, it is to be noted that we don't consider edges whose balances are 0. We have added comments to explain the code and data structures we have used to implement the Breadth First Search algorithm.

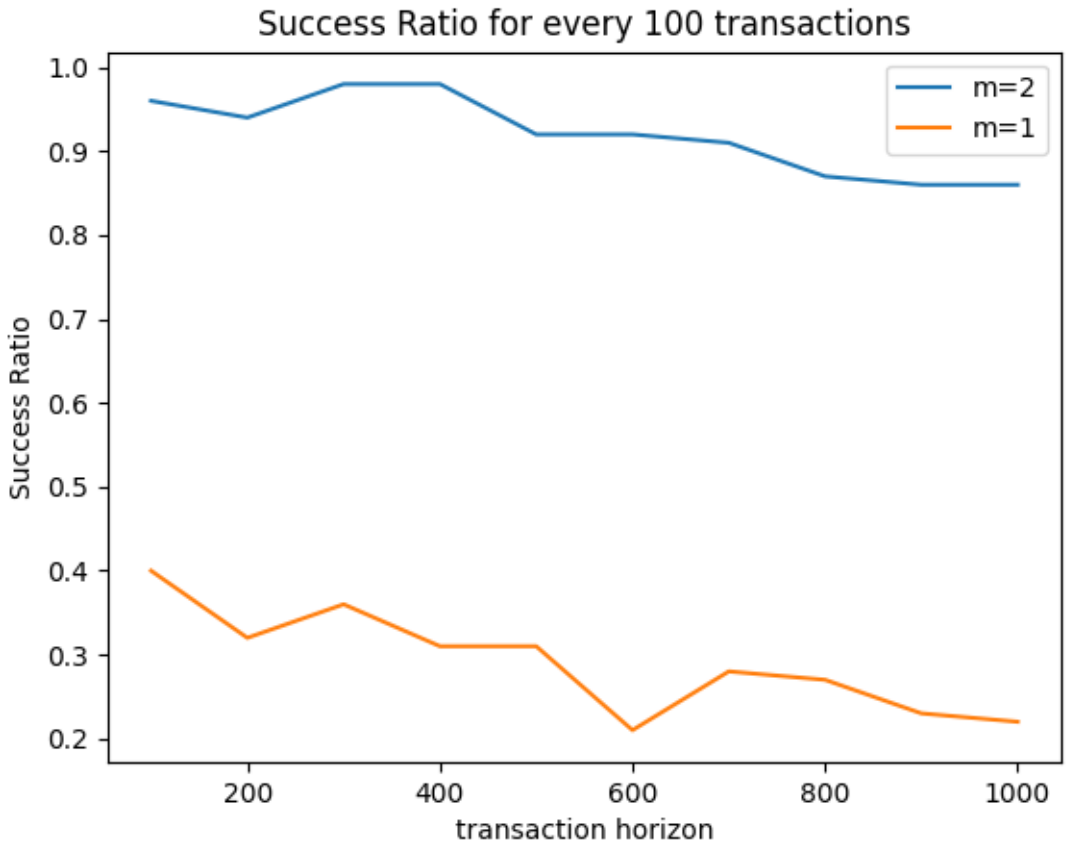
### 1.3 Client.py

The python file is used to create a simulation as instructed in the assignment. It generates a network with power law distribution containing 100 nodes and checks whether this graph is connected or not. It then creates contracts for registerUser and createAcc, setting up links between the clients based on the generated network. The assigned balance is obtained from an exponential distribution with mean 10 for every account. Finally a main loop create 1000 sendAmount transactions batched into groups of 100 and success ratio is calculated for every batch plotted in the results section.

### 1.4 Power Law Distribution

We use a library called `barabasi_albert_graph` from `networkx` to generate a graph. It implements Barabasi-Albert algorithm which is known to generate networks with power law degree distribution. It incorporates two important general concepts: growth and preferential attachment. Both growth and preferential attachment exist widely in real networks.

## 2 Results




In the above plot, m is a hyperparameter for the Barabasi-Albert algorithm which we have used to generate networks with power law degree distribution. Higher value of m corresponds to more densely connected network.

From the graph, we can observe that for both the networks, the success ratio roughly decreases over the transaction horizon. The reason behind this observation is that the initial transactions exhaust the balance on certain edges in the network. Thus, in the later transactions, BFS

algorithm is not able to find a shortest path between two nodes since most of the paths in the original network between the two nodes have now been exhausted by the initial transactions. This results in the failure of many transactions in the later stage leading to a decrease in the success ratio.

Moreover, it can be observed that the success ratio is higher in the case of a densely connected network (m=2) than a sparsely connected network (m=1). This is because naturally there will be more paths between any two nodes in a densely connected network than in a sparsely connected network.

| Transaction Horizon | Success Ratio |
|---|---|
| 0-100 | 0.96 |
| 100-200 | 0.94 |
| 200-300 | 0.98 |
| 300-400 | 0.98 |
| 400-500 | 0.92 |
| 500-600 | 0.92 |
| 600-700 | 0.91 |
| 700-800 | 0.87 |
| 800-900 | 0.86 |
| 900-1000 | 0.86 |

Table 1: Densely Connected Network

| Transaction Horizon | Success Ratio |
|---|---|
| 0-100 | 0.40 |
| 100-200 | 0.32 |
| 200-300 | 0.36 |
| 300-400 | 0.31 |
| 400-500 | 0.31 |
| 500-600 | 0.21 |
| 600-700 | 0.28 |
| 700-800 | 0.27 |
| 800-900 | 0.23 |
| 900-1000 | 0.22 |

Table 2: Sparsely Connected Network