CS765: Building a layer-2 DAPP on top of Blockchain

Team Composition:
Anushka (200050011)
Khyati Patel (200050102)
Shrey Bavishi (200050132)

March 2023

Contents

1 Introduction

A Dapp, or decentralized application, is a software application that runs on a distributed network. It's not hosted on a centralized server, but instead on a peer-to-peer decentralized network, such as Ethereum. Ethereum is a network protocol that allows users to create and run smart contracts over a decentralized network. A smart contract contains code that runs specific operations and interacts with other smart contracts, which has to be written by a developer. Unlike Bitcoin which stores a number (user balance in the form of UTXO), Ethereum stores executable code.
Dapp you are going to develop is accessed by different users where each user will form a joint account with other users of interest, i.e., with whom they want to transact. Each user specify their individual contribution (amount or balance) in the joint account. This way, the users will form a network, say user network Gu, where each user is a node, and a joint account between two individuals is an edge. It is not recommended to create a joint account with every other user whom one wants to transact. Therefore, Dapp allows two users to transact if there exists a path between them in the Gu. For example, if A has an account with B and B with C, then A can send the amount to C through B. Similarly, suppose C and D have a joint account. In that case, A can send the amount to D without having an account with D. If there are multiple paths between a pair of users, then the one with the least hop count will be selected, where the tie will be resolved with the first path in the obtained list of the path.

2 Implementation

Structure of the contract :

Contract
struct User
struct JointAcc
List of functions..

```
struct User {
    uint256 user_id;
    string user_name;
    bool exists;
    uint256 balance;
    mapping(uint256 => JointAcc) jointAccounts;
    uint256[] jointAccountsList;
    uint256 numJointAccounts;
}

struct JointAcc {
    uint256 user_id;
    uint256 balance;
}
```

Set of functions

- **registerUser (user id, user name)** : It will register the user and add it to the list of available users to transact.User id is just a number and doesn't have any public-private key implication.
- **createAcc(user id 1, user id 2,balance)** : Create a joint account between two users and keep a track of individual contribution to the joint account.This is equivalent to adding an edge to the network.
- **sendAmount(user id 1, user id 2)** : Transfer the amount (only whole number, no decimal) from one user to the other irrespective of having a joint account or not. If a user doesn't have sufficient balance to send the amount, it will reject the transaction and result in transaction failure. Otherwise, the transaction is successful.
- **closeAccount(user id 1, user id 2)** : Terminate the account between the two users passed as a parameter to the method call. This is equivalent to removing an edge from the network.
- **checkJointAccountExists(uint256 userId1, uint256 userId2)** : Used in the sendAmount function to check if transfer is possible between intermediate users
- **findShortestPath(uint256 fromUserId, uint256 toUserId)** : Helper function used in the sendAmount function to facilitate transfer of amount from a user to another by providing the shortest possible path if direct route does not exist.
- **removeJointAccount(uint256 userId, uint256 touserid)** : Function used in the closeAccount method for closing all the joint accounts of the user for which the account is being closed with any other user.

3 Network Topology

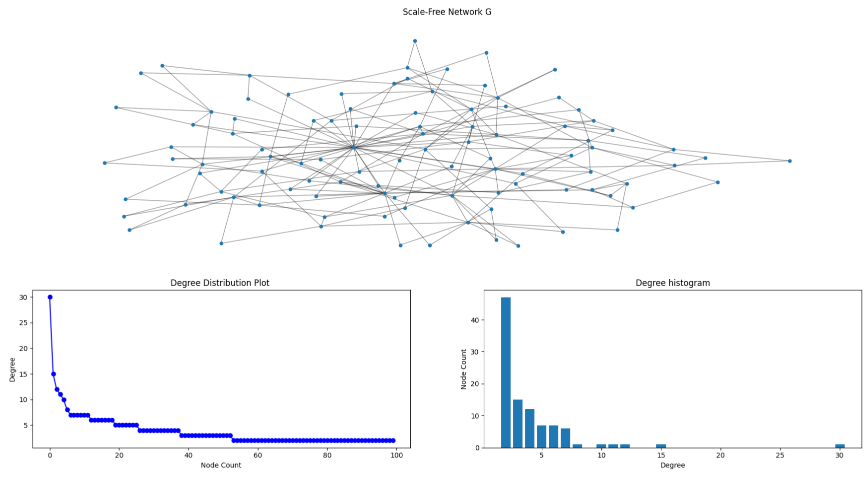

Figure 1: Connected network of 100 nodes following power law degree distribution and m=2

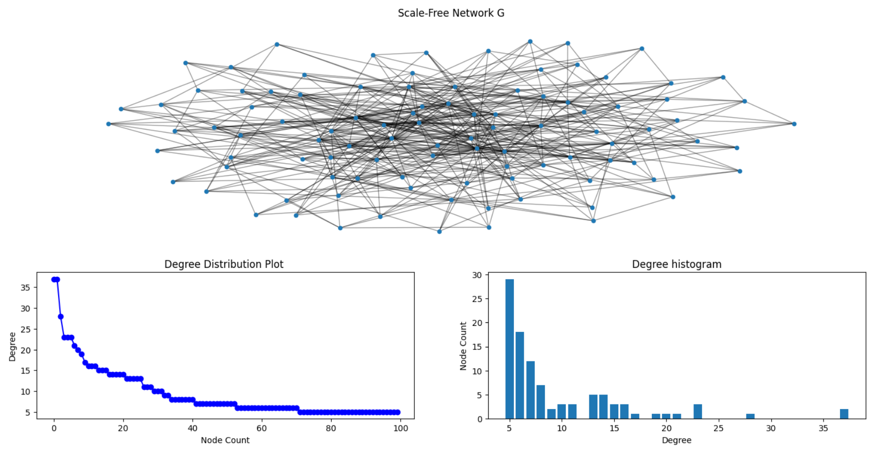

Figure 2: Connected network of 100 nodes following power law degree distribution and m=5

4 Results

- **RUN 1**: The ratio of successful transactions to total transactions for every 100 total transactions for a network with 100 nodes when m=2:

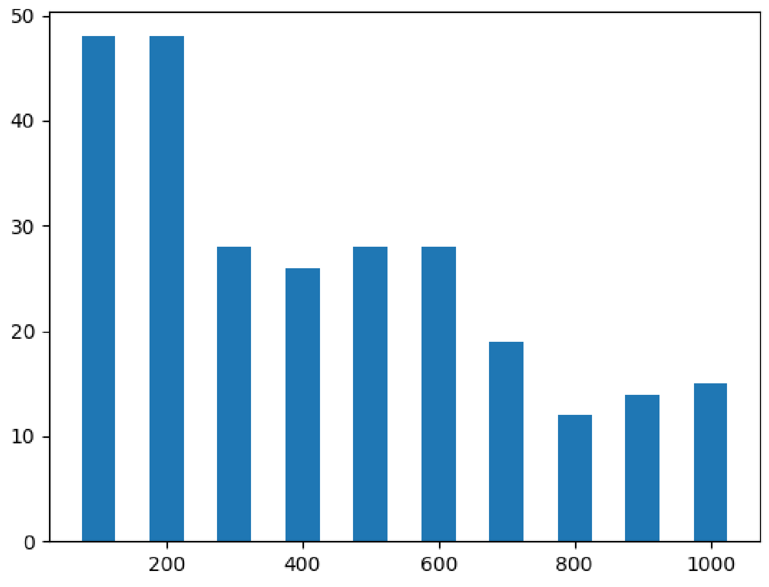

Figure 3: Run 1 for m=2

- **RUN 2**: The ratio of successful transactions to total transactions for every 100 total transactions for a network with 100 nodes when m=5:

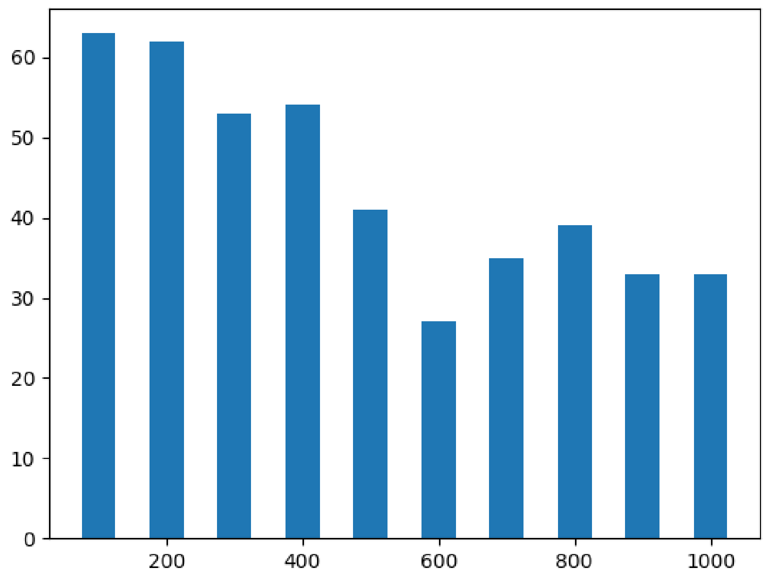

Figure 4: Run 2 for m=5

Observation As the total number of transactions executed increases,the number of successful transactions goes down.This is because as more and more transactions are executed, it is more likely that the user has exhausted its joint account balance with most of its peers.Thus, when there is a new request for a send amount operation, it is more likely to fail because of insufficient funds for the users present in the shortest path from user A to user B. We find out the shortest possible path based on least hop counts from user A to user B in the network.Everytime a transaction sending 1 unit from user A to user B is issued ,this path will be chosen irrespective of the inability to actually transfer coins due to insufficient funds.Since there are no alternative paths chosen ,repeatedly carrying transactions in this route will exhaust coins quickly and once exhausted will have no other option other than resulting in a failed transaction'1'1"1. It can also be observed that the number of successful transactions increases with m. This happens because: as m increases more connections are made among nodes hence decreasing the chances of balance of a user in a joint account going to 0 as it does not have to travel a longer path.