CS765 - INTRODUCTION TO BLOCKCHAINS, CRYPTOCURRENCIES AND SMART CONTRACTS

Report on Building a layer-2 DAPP on top of Blockchain

CHIRANMOY BHATTACHARYA
22M0744

SAI KUMAR ATLURI
22M0745

HEMANTH NARADASU
22M0777

Overview of the Smart Contract

State Variables

1. participants
   Number of users in the network.

2. adjList:
   A 2d array storing the adjacency list of each participant (user) in the network.

3. balance
   A hash map that stores the balance of a user in a joint account. The key to the hash map is a string of the form “id1->id2”, where id1 and id2 are the participants in the joint account, and the value reflects the balance of id1 in the joint account.

4. usernames
   A hash-map that stores the username corresponding to user_id.

Internal Helper Functions

1. uintToString
   Given an unsigned integer, return the integer as a string.

2. getKey
   Given two user ids, user_id1, and user_id2, return a string of the form “user_id1->user_id2”. This string acts as the key for the balance mapping, whose value represents the balance of user_id1 in the joint account between user_id1 and user_id2.

Public Functions

1. registerUser
   Register a user to the network. Push an empty array to the adjList, corresponding to this new user.

2. createAcc
   Creates a joint account between two users, user_id1 and user_id2. The adjacency lists of the users are updated to store the partner’s user_id. The balance in the account is stored in the balance hash map.

3. sendAmount
   Send an amount of money from a sender to the receiver. The BFS algorithm finds the least hop feasible path from the sender to the receiver. Edges (accounts) with infeasible balances are ignored to find a viable path. The transaction fails when no path can support the amount. This happens when

every path has one or more joint accounts whose balance is less than the amount involved.

4. closeAccount
   Close a joint account between two users, user_id2 and user_id2. The corresponding adjacency lists are updated by deleting the entry of the partner's user_id. The mapping in the balance hash map is also deleted.

Public Functions Used for Testing

1. jointAccounts
   For a given participant (user) user_id, return the users with whom the user_id has a joint account. The users are returned as a space-separated string of integers representing the user_ids of the partner.

2. balanceDistribution
   Returns the balances of all users across all the joint accounts. The balances are returned as a string of space-separated integers.

The Client Script

The adjacency list is generated locally. The adjacent nodes (users) are chosen from a random uniform distribution. Once a connected network is generated, the user accounts are created then the joint accounts are created. A joint account corresponds to an edge in the network. The degree distribution of the nodes follows a power-law distribution, while the joint account balance follows an exponential distribution. The explanation for the same is provided below.

Reason for Using Power-law Degree Distribution

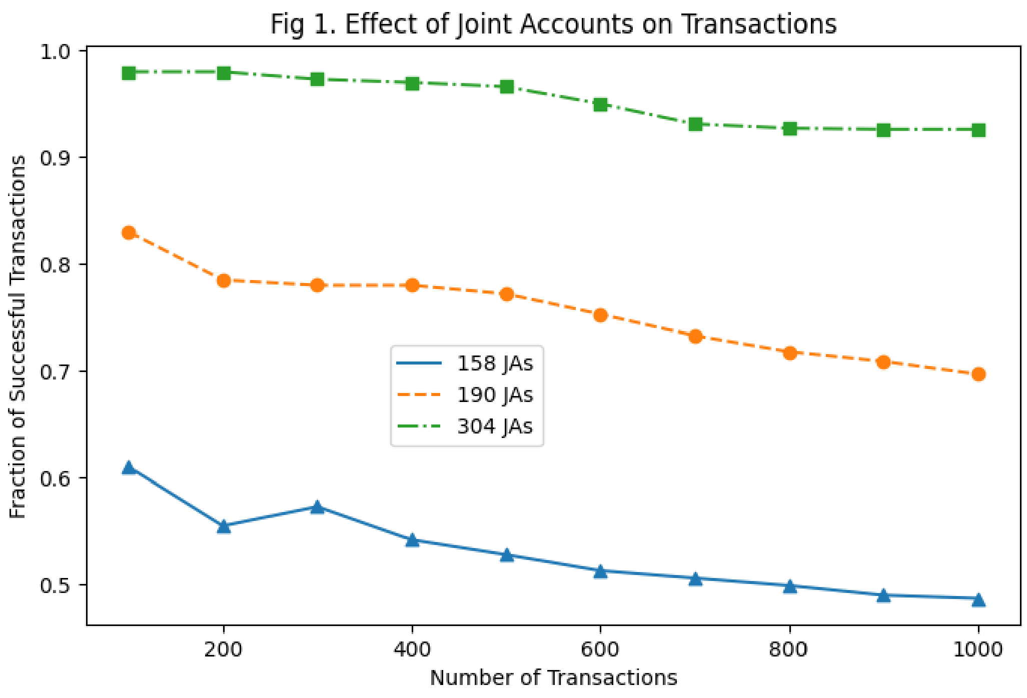
Power law distribution represents an 80-20 rule. In the context of the number of friends/associates of a person (user), 20 percent of users are connected to 80 percent of people. And 80 percent of the users are associated with 20 percent of the people. This means most users will have a joint account with a few others, while a few will be well-connected, i.e., connected to many.

Reason for Using Exponential Distribution for Joint Account Balance

The exponential distribution is used here because of its memory-less property. The amount chosen for a joint account should be independent and not affect the following account's balance.

Transactions are fired after all the joint accounts, i.e., the network, is ported. A transaction's status, i.e., success or failure, can be found by checking the logs and searching for the TransactionEvent record.

Analysis of the Obtained Results



The simulation is repeated by modifying the parameter of the power-law distribution. Modifying the power-law distribution parameter, the degree distribution of a node (user) changes. As seen in Fig 1, when the number of joint accounts increases, the fraction of successful transactions also increases. This is because more joint accounts imply more money in the system resulting in fewer failures.

The number of successful transactions decreases over time because the money flows from users with fewer joint accounts, implying low balance, to well-connected users with more balance. Because of the power-law distribution's 80-20 rule, nodes with low balance will be the most common. After the initial successful transactions, after the money has been transferred to the few rare well-connected users, any future transaction involving the same sender will fail unless it receives some money.