CS765 Spring 2023 Semester

Building a layer-2 DAPP on top of Blockchain

Due Date: 23:59 hrs, April 9, 2023

Note: Marks will be awarded for the points mentioned within the text. The marks have been indicated in square brackets (e.g. [1]).
In this assignment you will have to implement a Decentralized application (DAPP) performing the functionalities listed below. This assignment can be done in groups consisting of **at most 3** persons.

Unlike the previous assignment, this assignment mandate you to use **solidity** language for the Dapp implementation. If you include code from a publicly available source, then state the source in the code's comments. **Not more than 15% of the code should be taken from such sources.**

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* G_u, 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 G_u. 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.

Once a user A transfers an amount to the other user B along the edge, the balance that A contributes gets accumulated to the other end of the edge, as depicted in figure 1. Note that the combined balance of the channel remains constant. Furthermore, after processing a transaction from others, the node's combined balance remains constant. Only its contribution to the incoming and outgoing edge (account) will change. For example, for the transaction of 3 from A to C, incoming and outgoing edge (account) for B is A-B and B-C, respectively. B's combined balance before and after processing the transaction from A to C is 30. However, B's remaining balance (contribution) for the edge A-B changes from 17 to 20, and the edge B-C changes from 13 to 10.

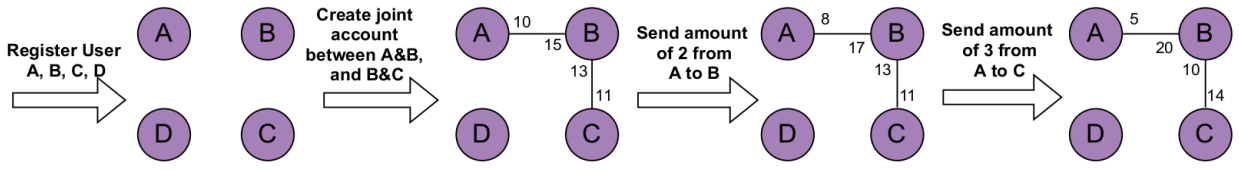

Figure 1: Small topology with four users A, B, C, D where A connected to B and B connected to C through a joint account, while D is isolated.

The Dapp is expected to contain the following functions. However, you are free to add your own functions for ease of implementation (if required). Once developed, you can test the correct working of your Dapp using the online compiler (remix.ethereum.org) mentioned in the useful link below.

Set of functions

- registerUser (use_id, user_name) : It will register the user and add it to the list of available users to transact. Please note that user_id is just a number and **doesn't have any public-private key implication**.[**10 marks**]
- 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. [**10 marks**] - 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**. For example, consider the last snapshot of the network in figure 1. If A tries to send 6 coins to C, then that transaction will **fail** because A doesn't have sufficient balance available in the joint account between A and B. Similarly, consider A's balance in the joint account is 15 and A tries to send 11 coins to C then B will reject the transaction as B doesn't have sufficient balance (it has only 10) in joint account between B and C. [**15 marks**]. 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. [**10 marks**] - This is equivalent to removing an edge from the network.

To test the Dapp, you have to set up an Ethereum node on your local machine using Truffle/Ganache. Please find the setup instruction file [2] in the useful links below. Deploy a Dapp (smart contract) you develop on the Ethereum node. Fire a transaction to call the individual functions (listed above) in the Dapp (call to each function [**5*4 = 20 marks**]). We suggest you write a python script to call the function and interact with the Ethereum node. Please find the demo script (client.py) to deploy the Dapp and call the function (send the transaction) in the assignment files. However, you can choose other languages of your choice for the same.

Your script should register 100 users and form a joint account between them such that the resulting network will be **connected** (unlike the one in the example above in figure 1) and follow the **power-law** degree distribution. Furthermore, the combined balance in the account for each pair should follow the **exponential** distribution with a mean of 10 and is distributed to two users involved in the account **equally**. [**20 marks**] Once the network is formed with the characteristics listed above, your script should call a stream of *sendAmount* transactions with a unit amount (1 coin) by randomly selecting a pair of the users in the network. You are expected to fire 1000 such transactions and report the ratio of successful transactions to total transactions after every 100 transactions and plot it [**5 marks**].

In your submission on Moodle, submit a single zip file (filename format: RollNo1_RollNo2_RollNo3.zip) containing:
1. Source code of the Dapp (smart contract) and the python (or any other language) script to deploy and fire the set of transactions.
2. README file with instructions for compiling and running.
4. A report detailing your findings appropriate reasoning behind it.
Marks: Proper commenting of code [**5 marks**], README file [**5 marks**]

Useful links:
`https://remix.ethereum.org/` - Online Solidity compiler
`https://docs.google.com/document/d/1IfIwdF6vhf4KYP1OYTwFmgqtdHO6nT7m0JaMH1xCIYo/edit?usp=sharing` - Set up Instructions