Software Systems Lab

Distributed Ledger
Design Document

Team 12: Gagan Madan (2013ME10015), Sarisht Wadhwa (2014TT10934), Rishab Goel (2013TT10958)

3rd August 2017

OVERVIEW

The aim of this project is to design a distributed ledger similar to Bitcoin. Code would be written primarily in Python. Public keys of all nodes would be stored in a Distributed Hash Table. Further, each node would contain address of all other nodes. This is needed in order to broadcast each transaction to all nodes currently in the network. Whenever a node comes online, it broadcasts its status to all nodes in the network, in order to ensure that further transactions are sent to this node. Also, some node would then share the key value pairs of the distributed hash table with this node.

CONTENTS

OBJECTIVES

1. Maintain a Distributed Hash Table to store all the public keys.
2. Simulate the 2-phase commit protocol among the three nodes that wish to do a transaction: Voting phase: Transaction is sent to three nodes (sender, receiver and witness) who vote if this transaction should happen, Decision phase: The initiator accepts the transaction if and only both the votes are yes.
3. Information sharing with all nodes: A general broadcast with digital signatures of each of the three participating nodes, amount transacted and reference to unspent incoming transactions would need to be done.
4. Verification by all nodes: Each node verifies the transaction by referring to the transaction list that it stores.
5. Handling Malicious Cases: Multiple simultaneous Broadcasts: Visual Synchrony to ensure order of transaction remains same for all nodes. Double Spend: Since order is defined we will always catch the double spent transaction by verifying in the history of transactions.

SPECIFICATIONS

Classes Proposed

Transaction
transaction_id: Unique id to identify a transaction
sender: Sender node id (account number)
receiver: Receiver node id (account number)
witness: Witness node id (account number)
amount: Amount to be sent
spent : bool variable indicating if the transaction is spent/used or not.

Node
addresses[]: An array of address for all nodes
online: A boolean indicating whether the node is online or offline in the system
transaction_history[]: A list of all transactions currently in the system
message_log : logs the data received to the node
l_clock: A scalar Lamport clock for virtual synchrony across nodes
dht : Distributed Hash Table containing the key-value for the public keys
online nodes : List of nodes that are online currently

Methods Used

Transaction
transaction(): Constructor, initialises a transaction with default values
spend : Makes a transaction non-spendable
sendable : Prepares data in a form of dict to be sent by the initiator

Node
send_message(node_address, msg_type, msg_value)
send_money(receiver_address, witness_address, amount, optional: input_transactions)
verify_transactions(sender_address, input_transactions, amount)
broadcast_message(msg_type, msg_value)
Online : make the node status online
distribute_keys(new_node)
Offline : makes a node offline
Create_trans_id : creates a transaction id for the transaction
Lamport_clock : updates the lamport clock of the node
Check_balance : returns the balance of the user
Verify : verifies that the user has the amount mentioned in the transaction
Send_history : returns a list of the transaction history of the user
Convert : update the current transaction history with received one, returns true

DHT (Distributed Hash Table)
initialize(): creates the key-value for the public keys of the users.
insert : inserts a key into the table
Get_value : returns the public key corresponding to the query

PROPOSED ARCHITECTURE

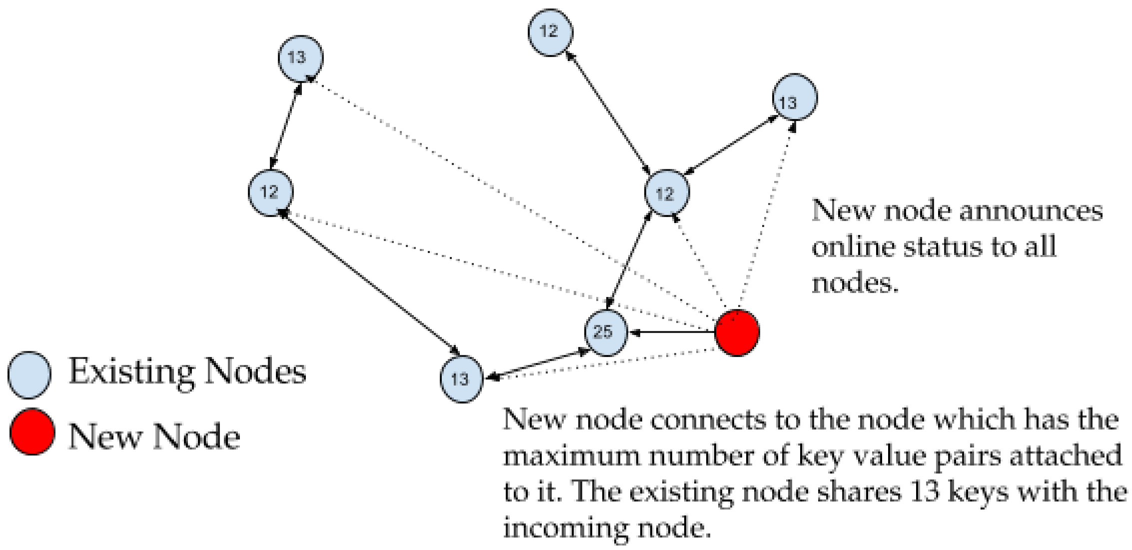
Communication Protocol

A new message would always first specify the message type being sent. The next message would be some string. The first message would inform the node about what to expect in the next message and how to parse it. Message type would be an integer denoting the following:

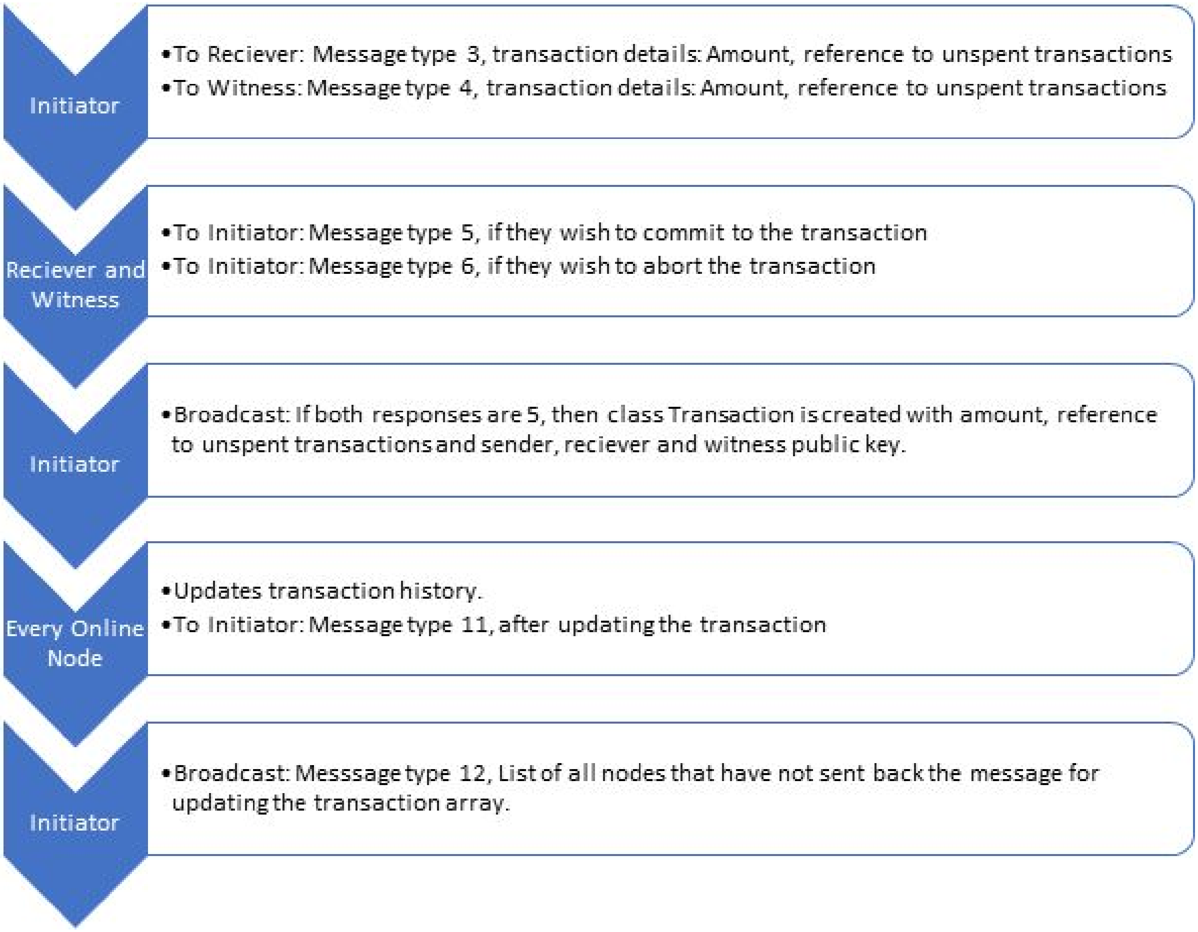
Message Type	Meaning
1	Node is now online
2	Update incoming node with transactions
3	Transaction Request to receiver
4	Transaction Request to witness
5	Commit (Response to message type 3/4)
6	Abort (Response to message type 3/4)
7	Broadcast Transaction
9	Send public key
11	Transaction updated (Reply from node to confirm update of transaction)
12	Nodes not updated list (Broadcast about which nodes have not yet been updated)
13	Ask for public key

New Node Protocol

Total nodes: 100
Nodes online: 7

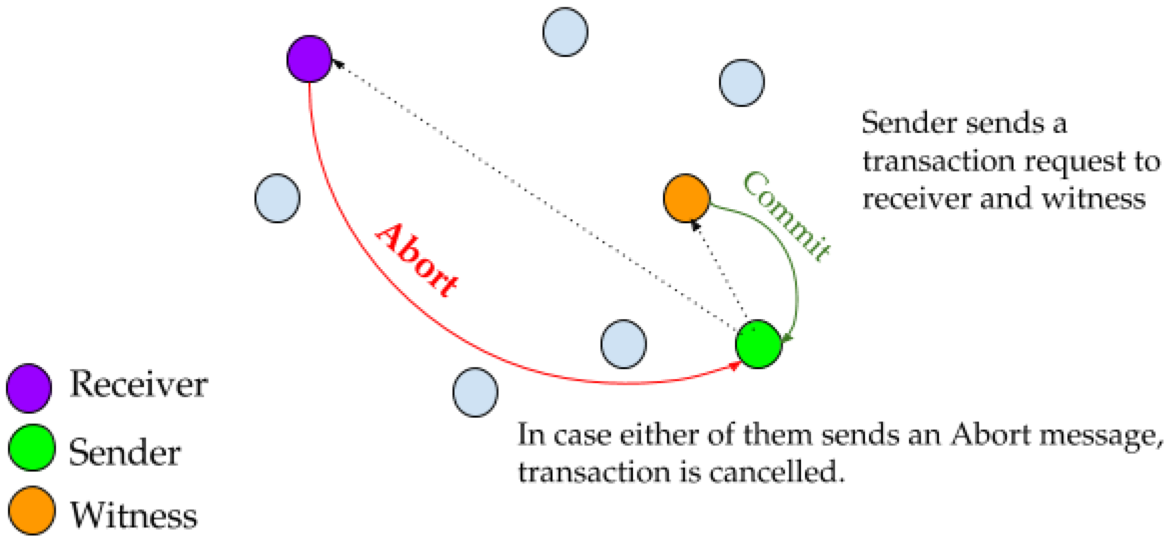


Transaction Protocol



Failure Protocol

If by chance a node goes offline during transaction, it won't be able to send back the message type 11 to the initiator and thus the node will be updated when it comes back online (list of nodes that have not been updated).



MILESTONES

1. “Hello World” Socket Program
2. Broadcast Messages (Messages sent to all nodes)
3. Maintaining a Distributed Hash Table
4. Single Transactions (2 Phase Commit)
5. Multiple Parallel Transactions
6. Visual Synchrony
7. Fault tolerance (Nodes going offline in the middle of transactions)