Patrick Cronin pjc2192
Damian Washel daw2213
Jonathan Eng jhe2215

Blockchain Wallet

We plan on implementing a distributed blockchain wallet. Our blockchain wallet will keep a ledger of all participants and their current balance.

What we need to do:

Verification

To do:

Mining operation
Create valid block
Mine valid nonce
Correction of following blocks after a change
Broadcasting updated blockchain to peers
Dealing with forks

Implement:

Implement: Implement: send the last hash code to all peers

Finding a nonce that creates a hash with a valid key (such as the first two digits being 0's) will be what verifies that a block is in the chain. We will find the valid hash by looping over possible nonce values until a hash value is found with the matching code.

Once a valid nonce is mined the newly minted block will be sent out to all the other peers on the network. Upon receiving a new block, peers will check to see if the received block's previous hash points to their current last hash. They will also verify that the new block has a valid hash and then update their blockchain by adding the block to the end.

When two chains mine a new block at the same time, we will run into the issue that we are trying to add two blocks at the same time. Because both blocks being sent out will have the same prev hash the other blockchains will only accept the first new block it receives and throws out the second block it receives. The second block is thrown out because the previous hash of the second block it receives will now be pointing to a block that is previous to the new last block. In order to reach agreement between all the block chains, there must be a vote and the chains in the majority will win out.

data structure for the transaction table

To do:

- Nonce
- Data: Ledger
- Prev Hash
- Hash
    - Verification key : 00

Implement:

We will create a linked list data structure with a built in hash function. To create a linked list in python, we will pass the hash of the previous block to the instantiation of the new block. The previous hash will be a data field of each block, thus the hash of each block will be dependent on the previous hash.

The block will also contain the data we want to store. In our case, the data we want to store is a ledger of all past transactions between peers. Each block will contain an exchange from peerA to peerB.

Peer-to-peer network

To do:

- Broadcast to peers
- Receive from peers
- Comparison of last hash in each blockchain against peers
- 1 tracker and at least 3 clients/peers (all on Google VMs).
    - Tracker maintains a list of peers
    - Every peer should be aware of any updates made to the list

Implement:

We will use a BitTorrent based P2P network. A tracker will maintain a list of peers that are currently in the network and add/remove peers that join/leave the network. The tracker will continuously ping its list of peers to verify that the peer is still active. If a node wants to join the network it must reference the tracker in order to obtain a list of current peers. With the list of peers, the initial action will be to request a copy of the blockchain. Then any time the blockchain is updated the updater sends the new block (and only the new block).

Interface

The interface of the video from MIT was very nice. Maybe we could imitate that interface.

One idea I had was that the wallet could be used for poker games. So a screen that looks like a poker table could be fun.

Working list of questions:

How to establish wallet?

What is the exact use of the tracker? Do we reference it every time we add a peer? Or can a new peer broadcast the current list when it joins the network.

Easy-sends alive? messages

How do we know a new entry is valid?

Do we need to implement being able to go back and change an entry? Don't

When dealing with forks, how do we decide which fork to continue with?/how do we decide the majority? Go with longest chain

Send the last hash code to all peers