A Blockchain Technology Simulation - RacketCoin

By
Nilay Pande 160070013
Sriram Balasubramanian 160070012

Note: Please use Racket version 6.9 or above (preferably 6.11)

Description:

We have implemented a cryptocurrency (RacketCoin) and simulated multiple transactions by several entities which are the miners and the nodes. The former will be able to mine and verify transactions, whereas the latter group will simply make the transactions. The RacketCoin protocol follows the Bitcoin protocol closely with very small differences. We have simulated several scenarios in which different miners have different computing powers, different connectivities with the nodes, different link delays with the nodes etc and have hence demonstrated the effectiveness of the distributed protocol. Our protocol like the Bitcoin protocol is robust against several cryptographic and non-cryptographic attacks.

Design:

The project mainly deals with two parts:

Implementing the cryptographic functions used in cryptocurrencies

A **digital signature** is a mathematical scheme for presenting the authenticity of digital messages or documents. A valid digital signature gives a recipient reason to believe that the message was created by a known sender, that the sender cannot deny having sent the message, and that the message was not altered in transit. The following functions deal with implementing a mechanism for signing digitally:

a. `(key-gen)` : A public key - private key pair generator for digital signatures.

b. `(dig-sign message private-key)` : Signing a message with your private key.

c. `(verify message sign public-key)` : Verifying if sign is indeed correct and not forged.

A **cryptographic hash function** is a one-way function where it is easy to calculate the hash of a byte string but difficult to calculate the inverse of the function. One such function is SHA 1(Secure Hash Algorithms). It is implemented as (sha m) in (openssl/sha1).

Now, we devise a “hard” puzzle for miners to crack. This constitutes the “proof of work” on which our protocol is based. If this puzzle is too easy, it would be easier for potential attackers to grow their own version of blockchain rapidly. A hard enough puzzle would be the following:
“ Give a sequence of bytes which when appended to the block being mined has a fixed number of zeros at the beginning of its hash”.
As the hash function has near random output behaviour, miner has to try different sequences of bytes to find such a hash. Hence, the answer to this puzzle is the required “proof of work” and time to find this depends upon miner’s computing power which cannot be monopolized by a single miner at all times.
This function is implemented as the following:

`(mine-for iterations block n)` : Tries a random sequence of bytes to append to the block and verifies if the first n bits of its hash are zeroes for the specified iterations.

Blockchain simulation implementation

Three classes are defined: miner, node, and global_events.

Miner:

The miner takes in new blocks broadcasted and checks if they are consistent with the previous blockchain. If no, they discard it. Else, they start “mining” the block. This implies finding a sequence of bytes as described previously. Once they get this byte sequence, they broadcast the block to the other miners.

If the miner hears that the block has already been mined, it simply drops its mining and accepts the new block, after verifying that it is indeed a valid block which is consistent with the RacketCoin protocol.

Node:

The node mainly does two types of transactions, paying and receiving money in the form of RacketCoin. After a transaction, it broadcasts its transaction to the miners

who validate the transaction and add it to the blockchain after validating and collecting it into a block.

Global_Events:

An event list which contains all events which are to be executed. It contains a priority heap which stores the events sorted by the time at which they occur. A timer is incremented and events in the events list are popped out one-by-one.

Sample of Inputs and Outputs

Sample Inputs are present in simulation1.rkt, simulation2.rkt and simulation3.rkt. Corresponding sample outputs are present in sample-simulation-trace-1, sample-simulation-trace-2, sample-simulation-trace-3.

simulation1.rkt is basic simulation demonstrating two miners and two nodes. All are connected to each other with link delays of 1 sec.

simulation2.rkt is the simulation having 4 miners and 2 nodes. There are 2 groups of miners (each having 2 miners) mining in competition with each other. All miners have equal computing power. After 2 confirmations, the nodes accept one of the versions of blockchain and subsequently rejecting the versions of miners who were late to mine.

simulation3.rkt again has 4 miners (groups of 2) and 2 nodes. However, this simulation demonstrates the effect of link delays on consensus in the distributed protocol. Though one miner group has much greater computing power than the other group, their version of blockchain gets rejected due to link delays.

Note that the simulation output will change every time simulation is run due to random number generation, random trials to mine etc.

Limitations

- We have not handled hash collisions between public keys, since the number of transacting entities are quite few.
- Joint payments (i.e a transaction in which more than two parties are involved) has not been implemented. However, we can simulate this using two party transactions.
- Even though our program is robust against double-spending attack and Denial Of Service attack, we haven't shown it explicitly in the simulation.
- We have used a relatively older cryptographic hash function i.e SHA1. These days cryptocurrencies use a newer and more secure hash function SHA256. The SHA1 algorithm is good enough for the purpose of our simulation and the migration to SHA256 can be done easily in the future.