Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

# 3D3 COMPUTER NETWORKS

## PROJECT 1

School of Computer Science and Statistics

(e-Report submission)

Submitted by:

**Rohan Taneja**
**19323238**
**Submission Date:** 11/03/2020

## Aim:

To establish a decentralized university/industry socializing network using Blockchain mechanism.

## A. Use Case:

My preferred use case is to implement a working peer-to-peer network in university and company networks which are usually centralized. This will not only help making the network onto peers' end but also keep user secure from data breach and give control to the user over their information. Securing personal information is the main goal of this network model. Which require only peer-to-peer network connection based on nodes connected on a network. Each having a unique identity. Every identity on this network is secure, authenticated and pseudonymous.

## B. Graphical Illustration:

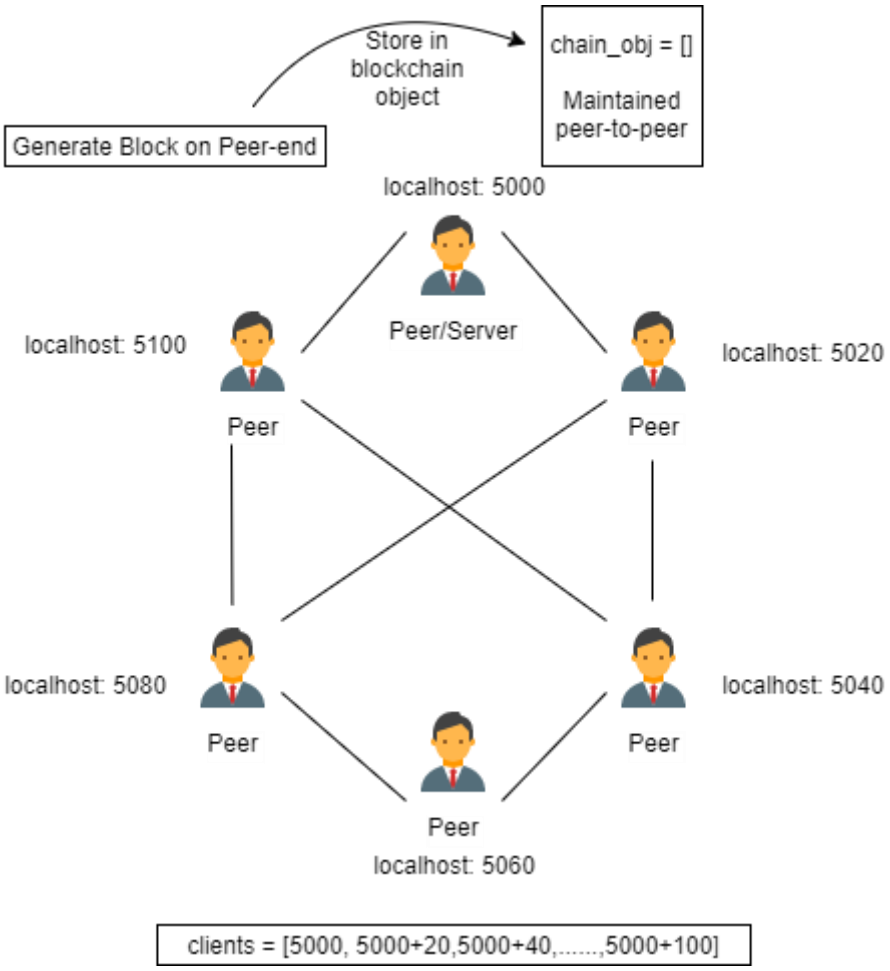


Figure 1. Illustrate working of my network model – made using draw.io

## C. Functional Description:

The basic functionality of blockchain has been implemented in module.py, client.py & server.py (python scripts).

**Module Python Script:**

It includes class object for Block and Blockchain. The block object generates and store hash value when called by Blockchain's constructor. And appends the data in the generated blockchain list. This is base for my project which can be manipulated to a vast usage.

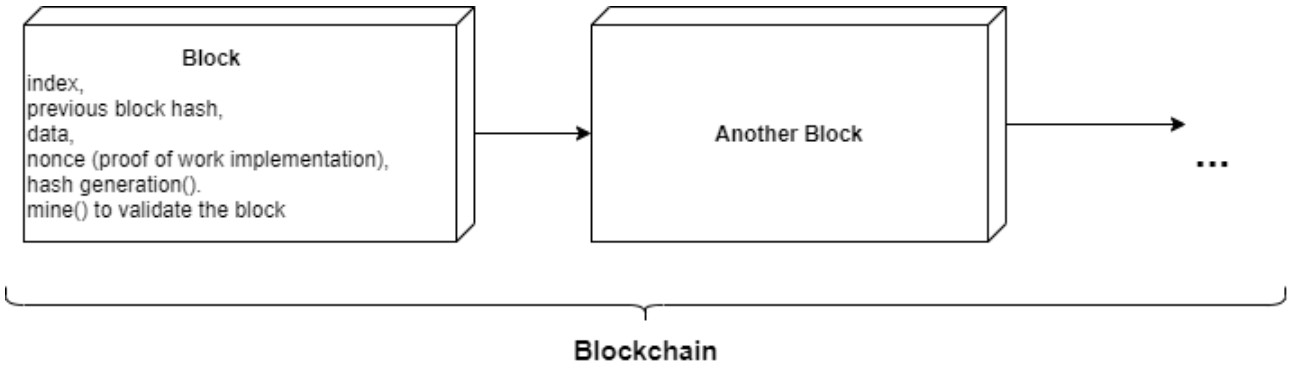


Figure 2. Implementation of my module – made using draw.io

Addition Info:

- hashlib used implement SHA256 hashing on the generated block

**Client Python Script:**

As the implementation is real-time and peer-to-peer functionality of Threading has been used in Python to keep the process asynchronous keeping multiple object event thread simultaneously.

- Transmission Thread Object – initializes with port, host, data, and socket (explained in low level implementation). The run () results in sending data message to network address as declared during client's execution.
- Transmission Validate Object – initializes with the socket connection with host and binding the socket. The run () is where the magic happens, as the implementation is based on proof of work. Therefore, the first 4 bits must be validated before operating on network. For '000' i.e. mined block will validate immediately and add itself to block chain. Whereas for other cases until it's achieved to a valid block generation with '0000' first 4 bits. For new connection established transmission functionality is provided in '1001' case.
- Data Transmission Object – serves for initial addition of block starting header with '1001' which is handled in previously defined object. Hence, the transmission takes place sending data after first addition to block has been done.

Addition Info:

- socket – used for low-level implementation of network,
- sys – to retrieve arguments from command line interface
- threading – to run multiple threads simultaneously
- json – to decode data from chain stored in json formatted object retrieving from server

**Server Python Script:**

As the implementation is real-time and peer-to-peer functionality of Threading has been used in Python to keep the process asynchronous keeping multiple object event thread simultaneously.

- Transmission Thread Object – initializes with port, host, data, and socket (explained in low level implementation). The run() results in sending data message to network address as declared during client's execution.
- Server Connection Object – Similar functionality to that of Transmission Validate Object. The incoming blocks are validated before addition. The purpose behind validation is to keep the identity unique of a block and keep in valid for the user who has ownership over the mined block.

Addition Info:

- socket – used for low-level implementation of network,
- sys – to retrieve arguments from command line interface
- threading – to run multiple threads simultaneously
- json – to encode data to json formatted object before transmitting to as chain to client

## D. Algorithm

**A. Basic Network Connectivity Algorithm:**

- Run **server.py** (master peer)
- Initializes the first block of the blockchain with null data, and hash pointing to itself.
- Hosts connection on UDP for master node.

1. Run **client.py _host_ _port_** (two arguments on run)
2. Calls socket connection on UDP which is connectionless initially for the master node.
3. Connects to the server (master peer).
4. Enter Data to asynchronously push to the network.
5. Module.py helps background working for block and blockchain implementation.
6. The block generates and get added to the network.

Stores hash to previous node and itself. Keeping the chain active on the network

- Now, connect more clients to check if the network established.
  (REPEAT step 1 – 6)

**B. Validation Algorithm:**

- Check if initial four bits are 0000.
  - Hence, mined and validates the block.
  - Adds to blockchain.
- Else
  - Verify if the client exists or not
  - And proceeds to mine keeping the block unique

## E. Networking Model:

As represented in the graphical illustration, the network is a peer-to-peer model. The low-level implementation includes network sockets which work on transport layer.

The implemented protocol in my model is **User Datagram Protocol (UDP)**. Which therefore keeps the block when generated local to the peer when mining it on the blockchain. When connection is established the socket transmits information the block validates.

As soon as handshake takes place between the peers the blocks generated appends to the blockchain.

## F. Other Considerations

### Security implementation:

- **Hash Encoded Block** – the generated block is hash coded to a unique hash using SHA256. Which keeps the block unique and the validated block when are added to the chain it stores generated hash as the token to that block.

### Communication and Functionality:

- **Handshake Mechanism** for communication between nodes in order to validate the network connection based on the authenticated token and identity.
- **Proof of Work:** for a nonce based approach, it works well as it involves the user contribution to network to lead the network accordingly. Hence, generating unique blocks at each mine.

## G. Implementation detail:

Initial approach towards building a network from grind is establishing a base network which would be localized. Which could be later hosted on cloud.

The connectivity is based on the UDP using network sockets. Which is handled asynchronously in user process using Thread objects.

Module implementation for Block and Blockchain are explained clearly in the functional description. Followed by the working of client and server. The blockchain generated is open distributed ledger i.e. anyone on the network can mine a block and append to the chain.

## H. Highlights

The unique selling point of my idea is that the model is based on public blockchain network which anticipates access to each node. But to keep it under control the network established need to have a restriction user role.

The functionality goal is to keep the network open to all users, with ease additivity of new node at any point. The network is user-controlled not owned by a group which would turn the open blockchain concept to consortium blockchain which has control on information retentivity on the network. Hence, the implementation application is vast.

## I. References:

1. draw.io for making the images used in the report
2. Ethereum Wiki on GitHub - https://github.com/ethereum/wiki/wiki
3. Develop a blockchain app from scratch in Python - https://developer.ibm.com/technologies/blockchain/tutorials/develop-a-blockchain-application-from-scratch-in-python/