Blockchain

CS346 Telecommunications & Networks

Uriyah Ann Ruiz
Henry Morgan
Kevin Peralta
Sarah Jimenez

What is blockchain?

To put simply, blockchain is a distributed database that maintains a continuously growing list of ordered records.

The term “blockchain” is strongly tied to the concepts of transactions, smart contracts, and cryptocurrencies.

Some popular blockchain based projects include Ethereum and Bitcoin

Implementation

Implementation

Block Structure

First, we must decide the block structure. We need to include the necessary items: index, timestamp, data, hash, and previous hash.

Block Hash

Each block must be hashed to keep the integrity of the data. SHA-256 is an authentication & encryption protocol used to verify all of the transactions.

Generation

In order to generate a block, we must know the hash of the previous block & create the rest of the required content given from the end user.

Storing

An array is used to store the blockchain. The first block is hard coded. This is called the “genesis-block”.

Block Structure & Configuration

```
class Block {
  constructor(index, previousHash, timestamp, data, hash) {
    this.index = index;
    this.previousHash = previousHash.toString();
    this.timestamp = timestamp;
    this.data = data;
    this.hash = hash.toString();
  }
}
```

Block Hash & SHA-256

```
void calculateHash = (index, previousHash, timestamp, data) => {
        return CrytpoJS.SHA256(index + previousHash + timestamp + data).toString();
}
```

The blockchain hash process may seem very short and simple in code, however with the integration of **SHA-256,** the internal process is much more complex.

If my only input was the character ‘a’, my output with the integration of SHA-256 would be ca978112ca1bbdcafac231b39a23dc4da786eff8147c4e72b9807785afee48bb

Generating a Block

```
var generateNextBlock = (blockData) => {
  var previousBlock = getLatestBlock();
  var nextIndex = previousBlock.index + 1;
  var nextTimestamp = new Date().getTime() / 1000;
  var nextHash = calculateHash(nextIndex, previousBlock.hash, nextTimestamp, blockData);
  return new Block(nextIndex, previousBlock.hash, nextTimestamp, blockData, nextHash);
};
```

The Genesis Block

The first block of every blockchain is called the *Genesis Block*. During implementation, this block is hardcoded into the chain.

```
var getGenesisBlock = () => {
    return new Block(0, "0", 1465154705, "my genesis block!!", "816534932c2b7154836da6afc367695e6337db8a9218943");
};
var blockchain = [getGenesisBlock()];
```

Validating Integrity

While working with a blockchain, at any given time, you must be able to determine whether a block or a chain of blocks is valid in terms of principle and integrity.

This is especially important when integrating a new block from other nodes into our chain. We must decide whether or not the block is valid or not.

To the right is a short code snippet of how we would be able to integrate checking the validity of a block.

```
var isValidNewBlock = (newBlock, previousBlock) => {
  if (previousBlock.index + 1 !== newBlock.index) {
      console.log('invalid index');
      return false;
  } else if (previousBlock.hash !== newBlock.previousHash) {
      console.log('invalid previoushash');
      return false;
  } else if (calculateHashForBlock(newBlock) !==
newBlock.hash) {
      console.log('invalid hash: ' +
calculateHashForBlock(newBlock) + ' ' + newBlock.hash);
      return false;
  }
  return true;
};
```

Conflicts

In the case of conflicts of blocks within two different chains, we will choose the chain with the most number of blocks.

For example, with chain A we have 4 blocks with one block with value 10. In chain B we have 7 blocks with one block with value 10. We choose chain B.

To the right is a code snippet of how we resolve such conflicts.