Blockchain Mid Term Solutions

Mayank Sharma, 160392

February 14 2019

1

- Increase the size of block:
  - Currently, the size of block is fixed, hence at max only a couple of thousand transactions can be included in a block.
  - Con: Larger blocks makes lesser number of miners/hashers to run as full nodes, hence causing the power to be centralized into some handful of pools. More data per block = more data to process for each node. This inevitably causes decentralization, and hence lowers the trust in bitcoin.
- Decrease the difficulty to mine a block.
  - This causes more number of blocks to be mined per hour, and hence increases the number of transactions.
  - Con: More number of forks may occur due to lower time to mine a block. It wastes hashing power and electricity too.

2

2.1 Output

The code can also be accessed at this link:
`https://github.com/mayanksha/blockchain/blob/master/midterm_programming/foobar.cpp`
Below code is compiled using

```
g++ -Wall -L /usr/local/lib/  -o foobar foobar.cpp -lssl -lcrypto
```

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAvVDOs4Ewxh6273y4kjDk
ivURFbbK1iQEWeGsEQip4P40LS3SRulE//+CzV3Pr5ZlpqCl+9sEKW0S8hFdjiVF
NjpWImWYhLevIl4Ne0CV6uiSOGN5a45uTXzsgpElTZUxCMzT6wTtjKVxcRhplOSP
pToLG8I5iR0fIdLOGudoBZNRODItGotCoUG9bSqrJi10rTjrOowDIjJg4itsECbP
UkfoaGQZJjTi2s2/Jl2BILX3LESyT0jeSOptpKmYbl+KBIl/qXYQtzj2laBUkx+6
eVn3v0/xMfCk374rjrDkLs3wb1BGXiwmHirIYYo/NuQHZ/uqgYIrHGe/r9muCq98
sQIDAQAB
-----END PUBLIC KEY-----

SHA-256 Hashed and then Base58 Encoded Public Key =
G78Kz4Ra5srKvNr1Fp1dqoEWuqnXaHGZabwKyxcyM2G1
```

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <string>
#include <vector>
#include <memory>
#include <assert.h>
#include <openssl/conf.h>
#include <openssl/evp.h>
#include <openssl/err.h>
#include <openssl/pem.h>
#include <openssl/crypto.h>
#include <openssl/sha.h>
typedef std::vector<uint8_t> uint8_vector_t;

static const char* base_58_chars = "123456789
    ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijkmnopqrstuvwxyz";
// Function to convert vector of bytes to its base58 representation
```

```cpp
std::string EncodeBase58(const std::vector<unsigned char>& vect)
{
  const unsigned char* start = &(vect[0]);
  const unsigned char* end = &(vect[0]) + vect.size() * sizeof(unsigned char);
  // Allocate enough space in big-endian base58 representation.
  int size = (end - start) * 138 / 100 + 1;
  // Skip & count leading zeroes.
  int zeroes = 0, len = 0;
  while (start != end && *start == 0) {
    start++;
    zeroes++;
  }
  std::vector<unsigned char> base_58_vect(size);
  while (start != end) {
    int carry = *start;
    int i = 0;
    // Apply "base_58_vect = base_58_vect * 256 + ch".
    for (auto it = base_58_vect.rbegin(); (carry != 0 || i < len) && (it != base_58_vect.rend()); it++, i++) {
      carry += 256 * (*it);
      *it = carry % 58;
      carry /= 58;
    }
    assert(carry == 0);
    len = i;
    start++;
  }
  // Skip leading zeroes in base58 result.
  auto it = base_58_vect.begin() - len + size;
  while (it != base_58_vect.end() && *it == 0)
    it++;
  std::string str;
  for(auto i = 0; i < (base_58_vect.end() - it); i++)
    str.push_back('0');
  while (it != base_58_vect.end()) {
    str += base_58_chars[*(it)];
    it++;
  }
  // Remove the starting zeroes from string too
  auto it2 = str.begin();
  while (it2 != str.end() && *it2 == '0')
    it2++;
  return str.substr(it2 - str.begin(), str.end() - it2);
}
void setup_context() {
  ERR_load_crypto_strings();
  OpenSSL_add_all_algorithms();
  OPENSSL_config(NULL);
  RSA_new();
}

void cleanup_context(){
  CONF_modules_free();
  CRYPTO_cleanup_all_ex_data();
  ERR_free_strings();
  EVP_cleanup();
}
int main() {
  // Setup the OpenSSL Context with various algorithms
  setup_context();
  int ret = 0;
  RSA *rsa = NULL;
  BIGNUM *bne = NULL;

  // set e
  bne = BN_new();
  ret = BN_set_word(bne, RSA_F4);
  assert(ret == 1);

  // generate key.
  rsa = RSA_new();
  ret = RSA_generate_key_ex(rsa, 2048 , bne , NULL);
  assert(ret == 1);
  // get rid of the bignum.
  BN_free(bne);

  // Declare a new Private Key and fill with random new value
  EVP_PKEY* privateKey = EVP_PKEY_new();
  assert (privateKey != NULL);
  // Use the declared rsa object to generate a private key
  EVP_PKEY_assign_RSA(privateKey, rsa);

  // dump public key in DER format
  uint8_t* derBuffer = NULL;
  auto derBufferLen = i2d_RSA_PUBKEY(rsa, &derBuffer);
  EVP_PKEY* publicKey = EVP_PKEY_new();
  assert (publicKey != NULL);
  // Use the rsa object to generate the corresponding public key for
    privateKey
  EVP_PKEY_assign_RSA(publicKey, rsa);
  assert (derBufferLen > 0);

  // Write the un-encrypted (not password protected) private key to
    stdout
  PEM_write_PrivateKey(stdout, privateKey, NULL, NULL, 0, NULL, NULL)
    ;
  // Write the public key to stdout
  PEM_write_PUBKEY(stdout, publicKey);
  unsigned char hash[SHA256_DIGEST_LENGTH];

  // Find the SHA256 Hash of the public key
  SHA256_CTX sha256;
  SHA256_Init(&sha256);
  SHA256_Update(&sha256, &derBuffer[0], derBufferLen);
  SHA256_Final(hash, &sha256);

  std::vector<unsigned char> temp;
  for(int i = 0; i < SHA256_DIGEST_LENGTH; i += 1)
    temp.push_back(hash[i]);
  std::cout << "SHA-256 Hashed and then Base58 Encoded Public Key = "
    << std::endl;
  // Base58 Encode the hash of public key
  std::cout << EncodeBase58(temp) << std::endl;

  free(derBuffer);
  cleanup_context();
}
```

foobar.cpp

3

The statement *"In bitcoin blockchain, double-spend attack is never possible"* is false. Double-spending attacks however uncommon can be possible due to the following two reasons:

- For the merchants accepting 0-confimation transactions, a double-spending is highly probable (called race attack). The person (say Alice) paying to the merchant for a service can send a valid transaction to merchant while also sending a conflicting transaction spending the coin to himself to the rest of network.

  Then, any malicious node (which can be controlled by Alice) can cause the correct transaction to not be included in the new block, whereas a double-spending transaction will get processed and be included in the next block by such a node.

- **51 % attack:** A single large enough node which controls 51% hash rate of the network can cause his own privately mined fork to grow to a large extent and then publishing his own mined blocks to the main chain. Moreover, since such an entity has large hashing power, he can mine blocks at a much faster rate thereby confirming the double-spending transactions faster than any heahtly node can. The moment of relief is that such an attack is highly improbable.

4 Selfish Mining

4.1

A selfish miner doesn't gets benefitted directly by mining selfishly, Instead, what he relies upon is wasting the hash/compute power of other miners to effectively gain advantage over them.

By broadcasting a block sometime later, he makes the other miners mine for the same block which has been found out by him. So, while he's trying to find a newer block, the others are just wasting their efforts on the previous block, thereby getting an unfair advantage over others.

4.2

Firstly, a globally synchronizable time is not achievable. Secondly, The miner who is mining the block is the master of the block at that time, so he can decide to forge a fake time on the new block. This will further cause problems with the blockchain.

5

5.1

Probability to solve hash-puzzle = p
Ways to choose k miners out of $n = \binom{n}{k}$
For k miners, prob. to solve hash-puzzle simultaneously

$$= \binom{n}{k} (p)^k (1-p)^{n-k} \tag{1}$$

5.2

Suppose in a certain time interval (t), we make n tries to find a block and out of that only k successes are achieved. The Probability to find a block (i.e. to solve a hash-puzzle) is p, and is fixed because the blockchain network changes the difficulty so that this remains constant.

Hence, in that time interval, the total number of blocks found out is equal to $\lambda = np$. Now, the distribution of finding a block in a large enough fixed interval follows the Binomial Distribution i.e. (Here, N_t is the number of arrivals in a time interval t)

$$P(N_t = k \mid p, n) = \binom{n}{k} (p)^k (1-p)^{n-k} \tag{2}$$

For a blockchain, the number n is sufficiently high, so that we can take
$n \to \infty$ and that $\lambda = np$, so that out $p \to 0$.

$$\begin{aligned} \lim_{n\to\infty} P(N_t = k \mid p, n) &= \lim_{n\to\infty} \binom{n}{k} (p)^k (1-p)^{n-k} && (3) \\ &= \lim_{n\to\infty} \binom{n}{k} \left(\frac{\lambda}{n}\right)^k \left(1 - \frac{\lambda}{n}\right)^{n-k} && (4) \\ &= \frac{\lambda^k e^{-\lambda}}{k!} && (5) \end{aligned}$$

This is the pmf for Poisson Distribution.

Now consider,
X_t : The time it takes for mining one additional block assuming that some block was mined successfully at time t. By definition of N_t, we have

$$(X_t > x) \equiv (N_t = N_{t+x}) \tag{6}$$

This means that the number of blocks which have arrived upto time t to $t + x$ is equal to N_t, or that no new block has been mined between time interval $[t, t + x]$. Now,

$$\begin{aligned} P(X_t \leq x) &= 1 - P(X_t > x) \\ &= 1 - P(N_{t+x} = N_t) \\ &= 1 - P(N_{t+x} - N_t = 0) \\ &= 1 - P(N_x = 0) \end{aligned}$$

The last step takes into fact that no new block was mined in time interval [t, t+x], and hence $N_x = 0$.

Also, λ in Poisson Distribution is the expected number of total arrivals. Take α to be the average number of arrivals per unit time. Hence, for a time interval x, we have a total of αx arrivals. From (5), we have,

$$\begin{aligned} P(X_t \leq x) &= 1 - \frac{(\alpha x)^0 e^{-(\alpha x)}}{0!} \\ &= 1 - e^{-(\alpha x)} \\ X &\sim \exp(x; \alpha) \end{aligned}$$

This is the CDF of exponential distribution, with an average time to mine a block of α. Hence, the time taken to mine a block in the next x time units has exponenttial distribution.

. **[Q.E.D]**

5.3

No, it's not possible to rely upon the timestamp provided by a miner to prioritize his block over someone else's. Due to network delays and congestion, a large amount of delay can generate between the time when a block was generated to the time when it was broadcasted. As a result, we can't have time synchronized over the world.

5.4

From part (b), the number of blocks which are mined in a certain time t follows exponential distribution. The number of blocks expected to be found in a certain time follows the Poisson distribution.

$$\begin{aligned} P(N_t \geq 6) &= 0.99 \\ &= 1 - P(N_t < 6) \\ 0.99 &= 1 - \sum_{i=0}^{5} P(N_t = i) \end{aligned}$$

Now, the parameter λ for Poission Distribution is the mean number of blocks expected blocks. For time x, we have,

$$\begin{aligned} \sum_{i=0}^{5} P(N_t = i) &= 1 - 0.99 \\ \sum_{i=0}^{5} \frac{(\alpha x)^i e^{-\alpha x}}{i!} &= 0.01 \end{aligned}$$

On solving, we get the value of $x \approx 130$ minutes

6 BoyCott possible?

Bitcoin has no mechanism to penalize any malicious user, thereby a scheme wherein we BoyCott some node is not possible. Even if such a scheme were possible, the malicious person can simply create a new wallet address by generating a new public/private key pair and using that to mine blocks. Blockchain also doesn't store any information related to the IP of miners hence an IP ban is out of the way too. Hence, boycotting any miner's chain can't prevent them from creating a new identity (priv/pub key pair) and start performing malicious behaviour again.

Instead, what blockchain relies upon for promoting normal behaviour is the game-theoretic possibilities which a node can use to maximize his profits. Each node picks a strategy to maximize its payoff by taking into account the other nodes' potential strategies. Every node is assumed to act according to its incentives, we can't split nodes into malicious or benign.

7 Mining Pools

The designer can change the hash puzzle problem from "Find the nonce such that the Hash belongs to the target space " to the new hash puzzle "Find the nonce and such that the hash of the digital signature belongs to a certain target space".

This setting automatically highly discourages a minig pool setup because calculating the digital signature requires the knowledge of private key of pool manager to all the nodes participating in the pool.

Any malicious person can steal the private key and use this to steal the money from the bitcoin wallet to which a block reward payment goes to on successful mining of a block. This way, a mining pool get split into a few trusted nodes only.

8 Solidity Program

```
pragma solidity ^0.5.0;

// Create a new Contract
contract MyFirstContract {
  // Declare balance as a uint (256 bit unsigned int)
  uint balance;

  // View function getBalance - View functions promise to not make
   any changes to state of contract
  // Return Value Type (uint)
  function getBalance() public view returns (uint) {
    return balance;
  }

  // constructor sets balance
  constructor() public {
    balance = 100;
  }
}
```

MyFirstContract.sol

```
const MyFirstContract = artifacts.require("./MyFirstContract.sol");

// Export the deployer of  MyFirstContract
module.exports = function(deployer) {
  // Deploy the MyFirstContract on Network
  deployer.deploy(MyFirstContract);
}
```

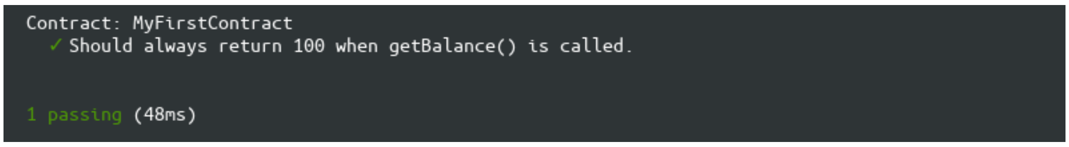
3_deploy_contracts.js

```
const MyFirstContract = artifacts.require("MyFirstContract")

contract('MyFirstContract', () => {
  it("Should always return 100 when getBalance() is called.", async
    () => {
    const firstContractInstance = await MyFirstContract.deployed()
    const balance = await firstContractInstance.getBalance()
    assert.equal(balance, 100,
      "Oops! Default balance wasn't 100");

  })
})
```

MyFirstContractTest.js



```
Contract: MyFirstContract
  ✓ Should always return 100 when getBalance() is called.


1 passing (48ms)
```

```
module.exports = {
  networks: {
    // Useful for testing. The `development` name is special - truffle uses it by default
    // if it's defined here and no other network is specified at the command line.
    // You should run a client (like ganache-cli, geth or parity) in a separate terminal
    // tab if you use this network and you must also set the `host`, `port` and `network_id`
    // options below to some value.
    //
     development: {
      host: "127.0.0.1",     // Localhost (default: none)
      port: 8545,            // Standard Ethereum port (default: none)
      network_id: "*",       // Any network (default: none)
     },

  },

  // Set default mocha options here, use special reporters etc.
  mocha: {
  },

  // Configure your compilers
  compilers: {
    solc: {
    }
  }
}
```

truffle–config.js

9 Pros and Cons

9.1 Advantages of Bitcoin over Ethereum

- Ethereum doesn't have a cap on number of coins, which means it has high inflation.
- Bitcoin uses UTXO model which has more scalability since transactions can be processed in parallel since they all refer to independent inputs.
- Bitcoin is much more widely accepted as compared to Ethereum. Many online wallets and merchants only allow payments via BTC, and not ether.
- Bitcoin is digital currency and has more market value than Ethereum.
- Ethereum's blockchain takes up A LOT of space. Over 1GB is added every month.

9.2 Advantages of Ethereum over Bitcoin

- Much faster transactions and their processing (12 seconds compared to 10 minutes for 1 block).
- Ethereum has a built in Virtual Machine and Ethereum is Turing Complete.
- Ethereum uses Account Based Model which provides advantages like larger space saving, simplicity and familiarity.
- Ethereum allows the use of smart contracts which has multiple advantages
- Ethereum is an open-ended decentralized platform which enables the creation of Distributed Applications over it's network.