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

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;

// Reference : https://github.com/bitcoin/bitcoin/blob/master/src/base58.cpp
// This is take from original source code for bitcoin, it's just a helper function
static const char* pszBase58 = "123456789ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijkmnopqrstuvwxyz";
std::string EncodeBase58(const unsigned char* pbegin, const unsigned char* pend)
{
```

```
21    // Skip & count leading zeroes.
22    int zeroes = 0;
23    int length = 0;
24    while (pbegin != pend && *pbegin == 0) {
25      pbegin++;
26      zeroes++;
27    }
28    // Allocate enough space in big-endian base58 representation.
29    int size = (pend - pbegin) * 138 / 100 + 1; // log(256) / log(58),
       rounded up.
30    std::vector<unsigned char> b58(size);
31    // Process the bytes.
32    while (pbegin != pend) {
33      int carry = *pbegin;
34      int i = 0;
35      // Apply "b58 = b58 * 256 + ch".
36      for (std::vector<unsigned char>::reverse_iterator it = b58.rbegin
       (); (carry != 0 || i < length) && (it != b58.rend()); it++, i++) {
37        carry += 256 * (*it);
38        *it = carry % 58;
39        carry /= 58;
40      }
41
42      assert(carry == 0);
43      length = i;
44      pbegin++;
45    }
46    // Skip leading zeroes in base58 result.
47    std::vector<unsigned char>::iterator it = b58.begin() + (size -
       length);
48    while (it != b58.end() && *it == 0)
49      it++;
50    // Translate the result into a string.
51    std::string str;
52    str.reserve(zeroes + (b58.end() - it));
53    str.assign(zeroes, '1');
54    while (it != b58.end())
55      str += pszBase58[*(it++)];
56    return str;
57  }
58  std::string EncodeBase58(const std::vector<unsigned char>& vch)
59  {
60    return EncodeBase58(vch.data(), vch.data() + vch.size());
61  }
62  void setup_context() {
63    ERR_load_crypto_strings();
64    OpenSSL_add_all_algorithms();
65    OPENSSL_config(NULL);
66    RSA_new();
```

```
67 }
68
69 void cleanup_context(){
70   CONF_modules_free();
71   CRYPTO_cleanup_all_ex_data();
72   ERR_free_strings();
73   EVP_cleanup();
74 }
75 int main() {
76   // Setup the OpenSSL Context with various algorithms
77   setup_context();
78   int ret = 0;
79   RSA *rsa = NULL;
80   BIGNUM *bne = NULL;
81
82   // set e
83   bne = BN_new();
84   ret = BN_set_word(bne, RSA_F4);
85   assert(ret == 1);
86
87   // generate key.
88   rsa = RSA_new();
89   ret = RSA_generate_key_ex(rsa, 2048 , bne , NULL);
90   assert(ret == 1);
91   // get rid of the bignum.
92   BN_free(bne);
93
94   // Declare a new Private Key and fill with random new value
95   EVP_PKEY* privateKey = EVP_PKEY_new();
96   assert (privateKey != NULL);
97   // Use the declared rsa object to generate a private key
98   EVP_PKEY_assign_RSA(privateKey, rsa);
99
100  // dump public key in DER format
101  uint8_t* derBuffer = NULL;
102  auto derBufferLen = i2d_RSA_PUBKEY(rsa, &derBuffer);
103  EVP_PKEY* publicKey = EVP_PKEY_new();
104  assert (publicKey != NULL);
105  // Use the rsa object to generate the corresponding public key for
       privateKey
106  EVP_PKEY_assign_RSA(publicKey, rsa);
107  assert (derBufferLen > 0);
108
109  // Write the un-encrypted (not password protected) private key to
       stdout
110  PEM_write_PrivateKey(stdout, privateKey, NULL, NULL, 0, NULL, NULL)
       ;
111  // Write the public key to stdout
112  PEM_write_PUBKEY(stdout, publicKey);
```

```
113    unsigned char hash[SHA256_DIGEST_LENGTH];
114
115    // Find the SHA256 Hash of the public key
116    SHA256_CTX sha256;
117    SHA256_Init(&sha256);
118    SHA256_Update(&sha256, &derBuffer[0], derBufferLen);
119    SHA256_Final(hash, &sha256);
120
121    std::vector<unsigned char> temp;
122    for(int i = 0; i < SHA256_DIGEST_LENGTH; i += 1)
123      temp.push_back(hash[i]);
124    std::cout << "SHA-256 Hashed and then Base58 Encoded Public Key = "
         << std::endl;
125    // Base58 Encode the hash of public key
126    std::cout << EncodeBase58(temp) << std::endl;
127
128    free(derBuffer);
129    cleanup_context();
130 }
```

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

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

Consider the exponential distribution with Probability density function (pdf)

$$\text{pdf} = \lambda \exp -\lambda x \tag{2}$$

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

9 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

10 Pros and Cons