Decentralisation of polling processes in developing nations: VoteBlocks Project Report

Sumuk Shashidhar - 12A
Adithya Mahesh -12A

Sri Kumaran Children's Home

March 16, 2021

Abstract

We put forward an idea and a prototype for an open-source, low-power, polling system built on traditional blockchain technology to decentralise the election process in developing nations in order to protect democracy and prevent voter fraud. Simple resource-efficient hashing mechanisms are used in order to ensure energy-efficiency but at the same time rely on the safety in scale approach to preventing attacks in the electoral process. Limitations of the voter-registration process and partial decentralisation problems are carefully examined and solutions are proposed for the same.

Keywords— elections, democracy, blockchain, decentralised, crypto

Sri Kumaran Children's Home, CBSE

Bangalore

This is to certify that

……………….........................

Of Class XII has successfully completed the project work titled -------------- -- under the guidance of ---------------------------------during the academic year ____________in partial fulfillment of _______________practical examination conducted by AISSCE.

Internal Examiner

External Examiner

Principal

Date of Practical Examination : _________________

Name of the Candidate: _______________________________

Registration Number: _______________________________

Examination Centre: _______________________________

Contents

1 Introduction

Democracy in developing nations is often challenged by electoral opacity. A lack of technological advancement/accessibility is a crutch and contributes to widespread voter fraud and voter suppression [1]. To combat this, we propose a decentralised plug-and-play polling system that allows the general public to take part in both aspects of the electoral process: organisation and voting. This also plays a psychological role as it allows the public to personally invest themselves in the process, thereby increasing voter turnouts and national democratic trust. VoteBlocks is the proposed prototype that is built on Python and Flask to ensure compatibility with a wide range of available hardware and minimal dependencies [2]. It consists of two parts that can be hot swapped: a frontend web application and a backend server. These can be run on the same machine or independently, thereby ensuring ease of use and greater control. It follows a traditional blockchain approach to recording of anonymous votes and allows any independent party to participate in the electoral process.

2 System Requirements

3 Project Flow

The voting flow consists of two groups of processes that are dealt with in stages: electoral and counting. We put forward stringent yet enforceable guidelines to ensure fair elections.

3.1 Electoral Process

3.1.1 Voter Registration

All voters must be registered at three separate centres with the same information. The logistical challenges are recognised, but this is crucial to keeping the process fair and democratic. Each centre must follow strict guidelines in the registration of voters, and ask for three separate pieces of identification and proofs of address. After the preliminary registration process, a synchronisation procedure must be conducted to remove inconsistent data and recapture missing or incorrect data. Care must be taken by registration centres to not store or leak personal information, but rather store hashes to ensure individuality while protecting voter anonymity. This is achieved by hashing user details immediately on capture.

3.1.2 Polling Process

On election day, voters must report to any of the available designated polling centres (the decentralised nature also removes many restrictions currently in place today that enforce the casting of votes to a singular region). Voters must report with their names, their pre-designated password and unique voter-id. Each voter must cast a vote as early as possible (see tallying process). Once the designed block-size limit is achieved, centres must mine the casted votes to add them to the network. These unconfirmed votes will be broadcast on the blockchain to allow public mining for purposes of trust distribution and decentralisation.

3.2 Tallying Process

Votes are tallied from a first-in first-out principle. For each voter-hash stored, a process is conducted to ensure that there are no other votes cast with a later time-stamp. If such votes exist, they are discarded immediately. This is in accordance with the blockchain double-spending problem solution [3]. A check is then performed to ensure that the corresponding voter-hash exists in more than two registration centre databases. If these tests pass, the vote is tallied towards the total for a candidate. The raw data of the blockchain with the specific voter-hashes is then broadcasted to the public to ensure transparency.

4 Blockchain Structure

The system is broken down into two major components. The block, and the chain. Each chain is made up a number of blocks, all interlinked with each other. Each block is populated with the transactions, i.e., votes in our model. When votes are cast, they are added to a list of unconfirmed transactions. When the block is to be mined, it will be put through a SHA-256 function numerous times in order to match a certain hash pattern. Once the right hash is formed using the nonce, it is added to the block. When combined, these entities form blockchains, which are hosted and co-ordinated by a number of Python Flask servers, to ensure redundancy. This ensures that the election process can continue with the support of a simple plug-and-play system, where nodes and servers can be swapped out while in operation to fix issues or improve performance. This is beneficial as it provides maximum redundancy while allowing industry-standard operational performance.

4.1 Structure of a block

Each block consists of the following elements essential to recording votes and identifying individuality of each voter. SHA- 256 hashing is used, where the data of the block is first input into a string, and then computed after being Unicode encoded.

- Index - To enumerate each block
- Transactions - The given transactions in each block (votes)
- Timestamp - The timestamp of each block's creation
- Previous Hash - The hash of the previous block to facilitate the blockchain
- Nonce - The number only used once. Used to supplement the rest of the data to generate a desired hash pattern

4.2 Structure of the chain

We discuss the structure of the chain as a combined product of its properties and functions as listed.

4.2.1 Difficulty property

The difficulty is a simple integer that determines how hard it is to mine a block. The higher this integer is set, the more difficult it is to mine. This is accomplished by using the number of leading zeros to the hash. A difficulty of 2 will ensure that there are two leading zeros for each accepted hash.

4.2.2 Genesis block property

The genesis block is the first mined block of that blockchain. Here, we mine it with a list of empty transactions and null (0) data, and we add it to the chain.

4.2.3 Last block property

A built-in function of the blockchain to retrieve the last added block. Useful in many use-cases to check timestamps and retrieve previous hashes.

4.2.4 Add block methodology

A simple function to add a block to the blockchain which requires only the block object and the proof of work. The block is verified by checking the proof of work to ensure its validity, and ensuring that the previous block hash of the blockchain, and the previous hash property of the submitted block match.

4.2.5 Proof of work function

A simple function that tries different nonce values to satisfy the hash pattern requirements set by our difficulty criteria.

4.2.6 Proof of work function

A simple function that tries different nonce values to satisfy the hash pattern requirements set by our difficulty criteria.

4.2.7 Proof of work function

A simple function that tries different nonce values to satisfy the hash pattern requirements set by our difficulty criteria.

4.2.8 Proof of work function

A simple function that tries different nonce values to satisfy the hash pattern requirements set by our difficulty criteria.