A Blockchain based Distributed E-Voting application implemented on Hyperledger Fabric.

GitHub: HyperVoter

Harpreet Virk, Vir Jhangiani, Varsheel Deliwala
CS - 2361 - Blockchain & Cryptocurrencies
Prof. Mahavir Jhawar
May 11th, 2020

Abstract

Existing voting systems, especially for elections, are usually carried out in a way where the participating members - Election Commission, Candidates, and Voters don't trust each other completely. This leaves the system vulnerable to attacks not only from outside adversaries, but also by the system participants themselves.

Online voting systems have often been considered as potential alternatives to the traditional methods of voting. In this paper, we will propose to devise a Blockchain based E-Voting System that will try to address these issues of security, privacy, convenience and trust.

1 Introduction

Traditional paper-ballot based voting systems are vulnerable to malpractices such as election rigging and forced ballot. The existing voting system relies on third party tally agents to count votes resulting in inaccuracies in the tally and even misreporting. Moreover, the fact that the voter turnout has decreased in the recent years suggests that these may not be the most effective and convenient systems going forward.

E-Voting systems, a popular alternative, have often been rejected in practical scenarios over concerns regarding lack of security, privacy, and trust due to their susceptibility to hacking. Malicious users can manipulate the election results by tampering with the way in which the identity of voters is verified, or the way in which the votes are submitted and counted.

In our E-Voting system, we aim to tackle these issues by ensuring the following:

- Every voter should be approved to vote before casting their ballot.
- The identity of every voter remains anonymous throughout the process. (Privacy Guarantee)
- No one should be able to cast their vote more than once in the same election. (Double Spend Problem)
- Votes should not be modifiable once cast.
- Election results should be publicly visible to the entire network to ensure transparency, with voters allowed to verify if their votes were correctly cast. (Voter's identity will still be anonymous)

Naturally, Blockchain can help us provide solutions to these problems with its distributed and decentralised public ledger, peer-to-peer consensus, immutability, and verifiability, which are ideal for a reliable voting platform. It guarantees that malicious parties cannot influence the voting procedures, thereby ensuring a secure and transparent E-Voting system.

2 Architecture

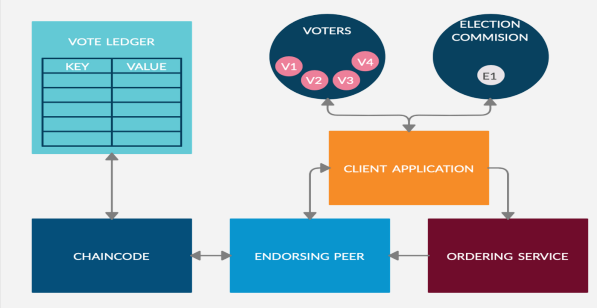
In this project, we will be using Hyperledger as the incubator for blockchain technology, with Hyperledger Fabric providing the basic framework for the architecture layer of the application.

With a permissioned ledger like Hyperledger Fabric, we can ensure that only verified voters are

able to participate to vote in the network, while ensuring that anonymity of the voter is maintained.

Our model will be built using a single client application with multiple interfaces/dashboards for different types of users.

One endorsing peer will manage the execution of chaincode inside its docker container, and one ordering node will be responsible for ordering the transactions.



2.1 Network Configurations

This application uses the sample network 'basic-network' which bootstraps the following instances and instantiates them on docker containers:

- 1 Orderer
- 1 Certifying Authority
- 1 org (org1) maintaining 1 peer (peer0)
- 1 CouchDB
- 1 CLI

3 Chainstate

The Chainstate Assets comprise of Vote Objects with the following attributes:

KEY	VALUE
voteID	{ownerId, hasVoted}

The VoteId is sequentially assigned whenever the EC registers a new voter and creates a new vote object for them. The ownerId is set to the voters's voterId, and has_Voted is set to false to signify that the vote object has not been used by the voter.

When a voter casts their vote to a candidate of their preference, the ownerId is changed to the candidate's Id and has_voted is set to true to signify that the vote has been casted and cannot be reused.

An example Chainstate can be seen in the following figure, where three digit ownerIds signify voterIds and four digit ownerIds signify candidate IDs:

VOTE OBJECT

KEY = {Vote_ID}	VALUE = {Owner_ID, has_Voted = False}
{13}	{434, False}
{14}	{767, False}
{15}	{889, False}
{3}	{6767, True}
{12}	{8664, True}
{2}	{1221, True}

4 Chaincode

The chaincode comprises of the following functions:

4.1 initLedger(ctx)

This function is invoked after the chaincode is installed and instantiated on the peer for the first time.

4.2 createVoteObj(ctx, sendTo)

The EC invokes this function to assign new vote objects for new voters. For every new vote object:

- A sequential vote id is assigned,
- OwnerId is set to the voterId of the voter,
- hasVoted is set to false.

4.3 setEndTime(ctx, endTimeVal)

The EC invokes this function to set an end time for the election. It takes the election duration in minutes as an input, and adds that to the current time to define the end time.

Before the election end time is set by the EC, voting is turned off so that voters cannot cast their votes. After the end time is set, voters are allowed to cast their votes.

However, until the end time elapses, all query functions are disabled.

4.4 sendVoteObj(ctx, voterId, sendTo)

The voters invoke this function to caste their votes to their preffered candidatdes. When the function is invoked, it first checks:

- If the voting has not started yet, it returns without casting.
- If the current time is greater than the end time, it returns without casting.

Then, it searches for the vote object which is owned by the voter. If no vote is owned by the voterId, the vote must have already been cast and the voter is trying to double spend. An error is returned, and the vote is not cast.

If the vote object is found, it is fetched from the chainstate. The owner id is changed to the candidate id, and the hasVoted field to set to true.

4.5 getResults(ctx)

This function can be invoked by anyone to get the total counts of votes that are sent to each candidate. However, it can only be invoked once the voting has ended.

4.6 voterTurnout(ctx)

Any user can invoke this function to check the current voter turnout. It will display the following:

- The number of Registered voters.
- The number of voters who have already cast a vote.
- The current voter turnout percentage.

It runs a loop through all the vote objects on the chainstate. If the hasvoted variable of the vote object is true then it increments number of votes cast by 1. Then it just divides number of votes cast by number of registered voters to find voter turnout percentage.

4.7 queryVote(ctx, voteObjId)

Any voter can invoke this function. This function queries a vote object by searching for the voteId in question. If voting has not started yet then it does not allow you to query a vote. If the vote has not been cast yet then it will not allow you to query the vote. If such a vote ID does not exist yet then it will simply return an error massage saying that this vote object does not exist.

It takes the key value voteId and returns the vote object attributed to that voteId only once the vote has been casted. We created this function so that a user can immediately query if their vote has been sent to the correct candidate right after it has been cast.

4.8 queryAllVote(ctx)

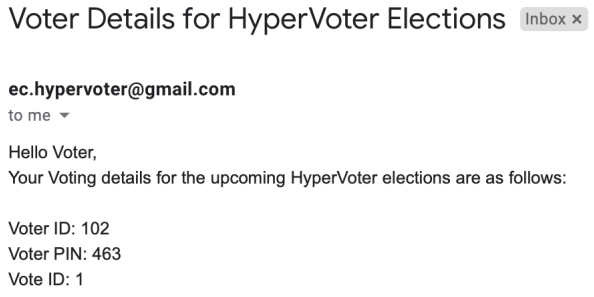
Any voter can call this function to view all the votes that have been cast. It does not let you call this function untill voting has ended. It loops through all the vote objects on the chainstate and displays them only if they have been cast.

5 Voting Procedure

When the Election Commission wishes to begin the elections, it does the following:

- EC Registers an EC account on the application, and logs into the EC dashboard.
- Register voters by providing their e-mail ids. On submitting, a vote ID, a voterId and a random PIN are generated, which are then sent to the voters e-mail id.
- Register candidates by providing their names.
- Set election end time to begin elections.

When a voter is added by the EC, they receive an email from the EC which contains their voterID, voter PIN and the vote ID as follows:



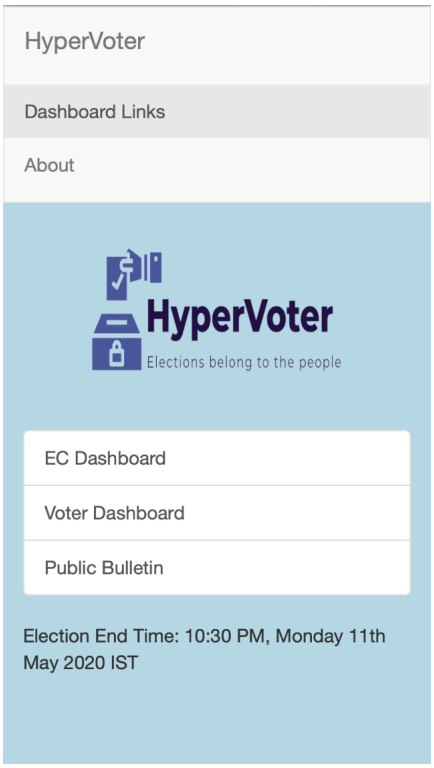
Once the elections begin, the voters can cast their votes as follows:

- Register their voter account using the details provided in the email.
- View the candidates list on the voting page, and note the candidate Id of the candidate that they wish to vote for.

- Enter the candidateId in the space provided to cast their vote.

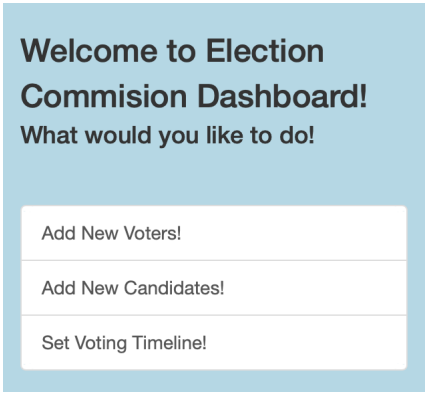
6 Client

The Election Commission (EC) user and Voter users both interact with the Blockchain interface using the front end web application. It has three dashboards:



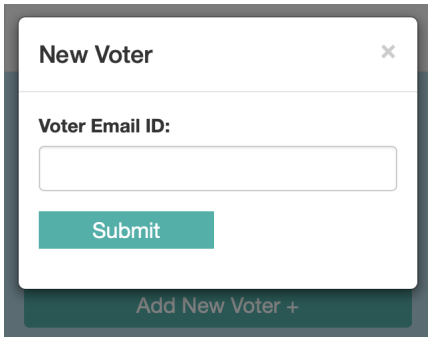
6.1 EC-Dashboard

The access to Election Commission Dashboard requires authentication with a pre-set EC Id and PIN, after which the EC user is logged in:

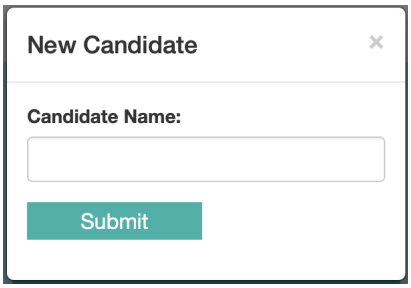


Here, the EC has access to the following functionalities of the application:

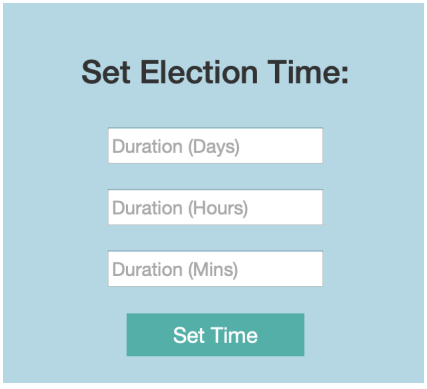
6.1.1 Add new Voters!



6.1.2 Add new Candidates!

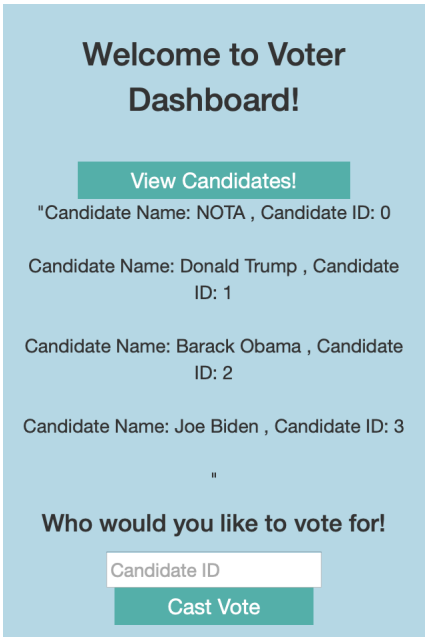


6.1.3 Set Election Time!



6.2 Voter-Dashboard

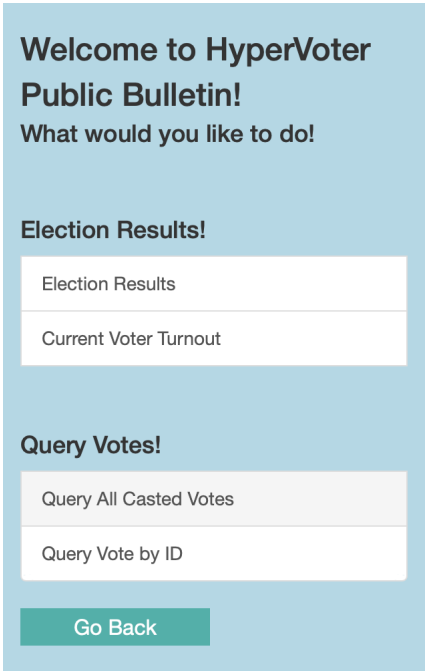
The voters dashboard requires authentication using the voter credentials that are sent to their email id (when EC adds the voter). On logging in, they get access to the voting page:



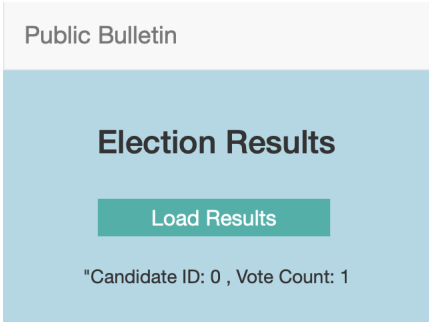
Here, they can view the registered candidates, and cast their votes to their preferred candidates.

6.3 Public-Bulletin

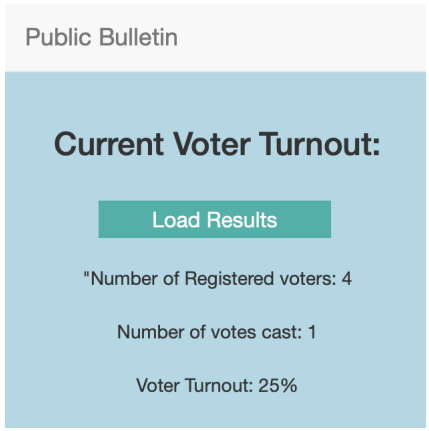
The public bulletin can be accessed by both the voters and the EC. It allows the user to view election statistics. The Election Results and Query to view all casted votes can be accessed only after the election end.



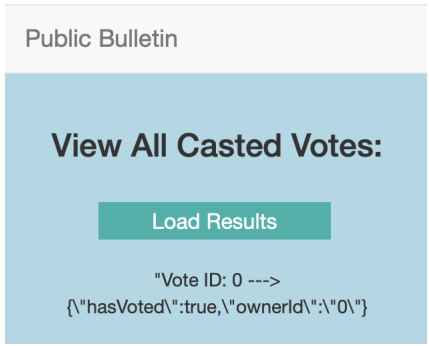
6.3.1 Election Results



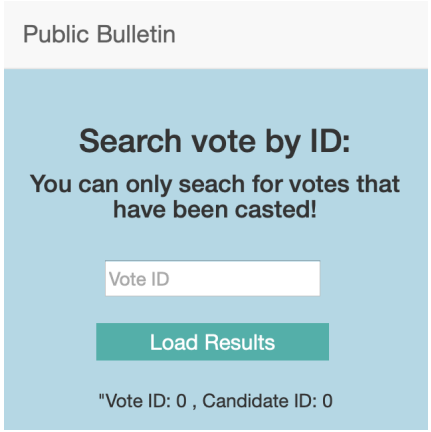
6.3.2 Current Voter Turnout



6.3.3 Query All Casted Votes



6.3.4 Query Vote by Id



7 Technologies Used

- Hyperledger Fabic
- Docker Containers
- CouchDB
- node.js
- Express.js
- HTML 5
- CSS 3
- JavaScript

8 Usage Instructions

8.1 Prerequisites

1. Download HyperLedger Fabric v1.4.6.
2. Download HyperVoter repository and merge its contents with fabric-samples directory.

8.2 Network Setup

To install, instantiate, and invoke the chaincode, do the following:

```
$  cd fabric-samples/hypervoter
$  ./teardownHyperVoter.sh
$  ./startHyperVoter.sh
```

If ./sh files have permission error (mac OS):

```
$  chmod u+r+x ./file_name.sh
```

8.3 Viewing Chaincode Logs

```
$  docker logs -f dev-peer0.org1.example.com-hypervoter-1.0
```

8.4 Running the Application

1. Change working directory:

```
$  cd fabric-samples/hypervoter/javascript
```

2. Install Application Dependancies:

```
$  npm install
```

3. Run Application

```
$  node app.js
```

4. To view the frontend, go to your browser and lauch http://localhost:5000

9 Conclusion

We designed and implemented a system for voting, based on blockchain, with the following outcomes.

- Transparency in vote counting to reduce malpractices.
- Anonymity of Voters is maintained, whilst allowing them to self-verify that their vote went to the desired candidate using Vote_ID at the same time.
- Does not allow double spending of votes.
- Eliminates the possibility of error in vote counting.
- Decentralised networks enhance trust in the system.
- Increase in vote count, as the process is made more convenient.
- Reduction in the election costs.

A good voting system should be able to help conduct elections in a secure, reliable and transparent manner. Our project provides insight into this problem of building a voting system that can be trusted to hold fair elections.

In future work, we would like to work on including different models of holding elections such as Multi-Candidate System, and the Swiss Proportional Representation. Moreover, we would also like to incorporate different methods of vote counting such as dual EQ and list-votes to allow voting in a multi-party system.